Algorithm

Merge Sort

Example | Visual

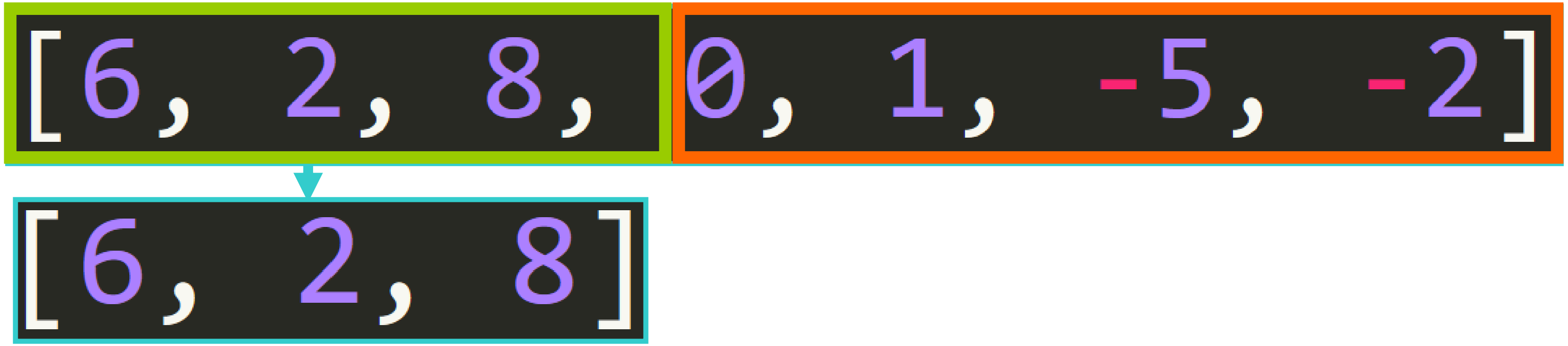
Merge Sort

```
[6, 2, 8, 0, 1, -5, -2]
```

Merge Sort

```
[6, 2, 8, 0, 1, -5, -2]
```

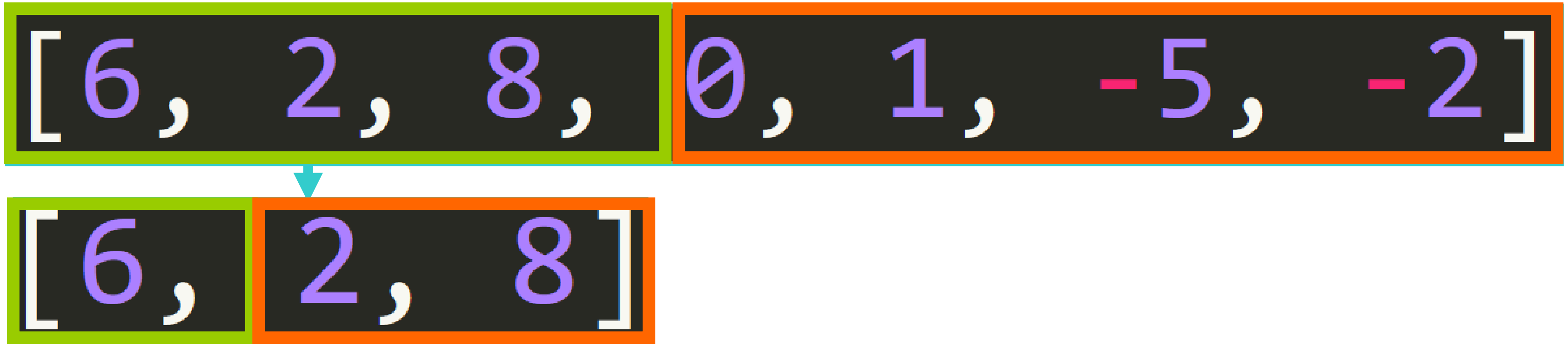
Merge Sort



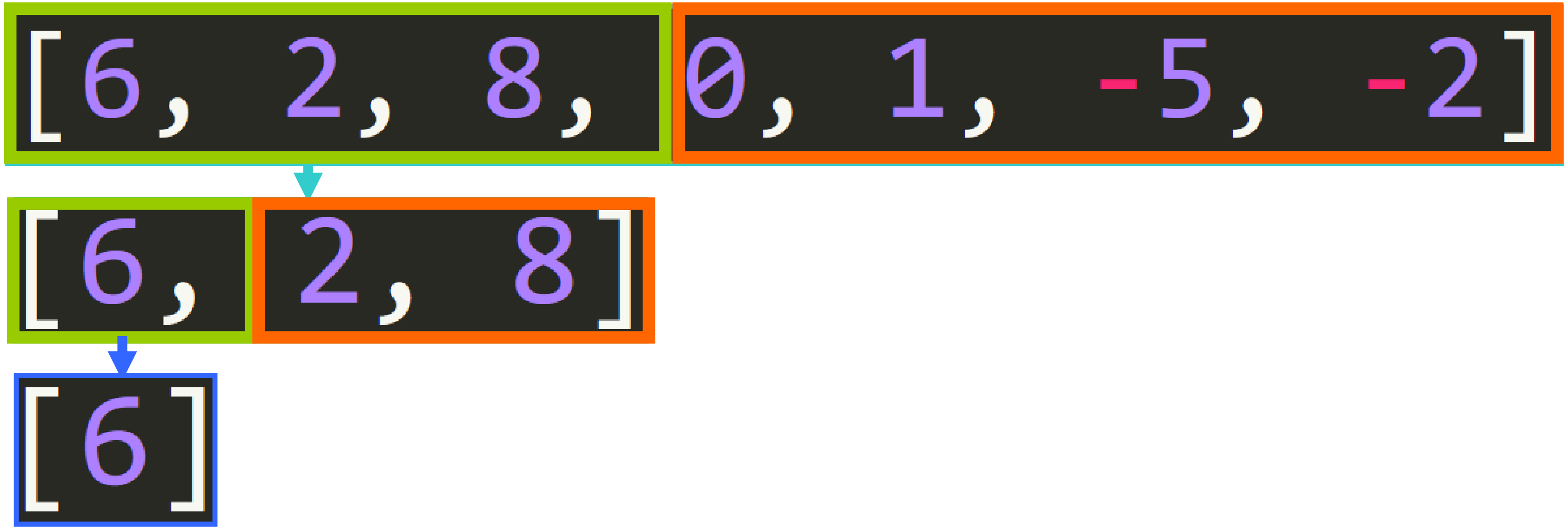
```
[6, 2, 8, 0, 1, -5, -2]
```

```
[6, 2, 8]
```

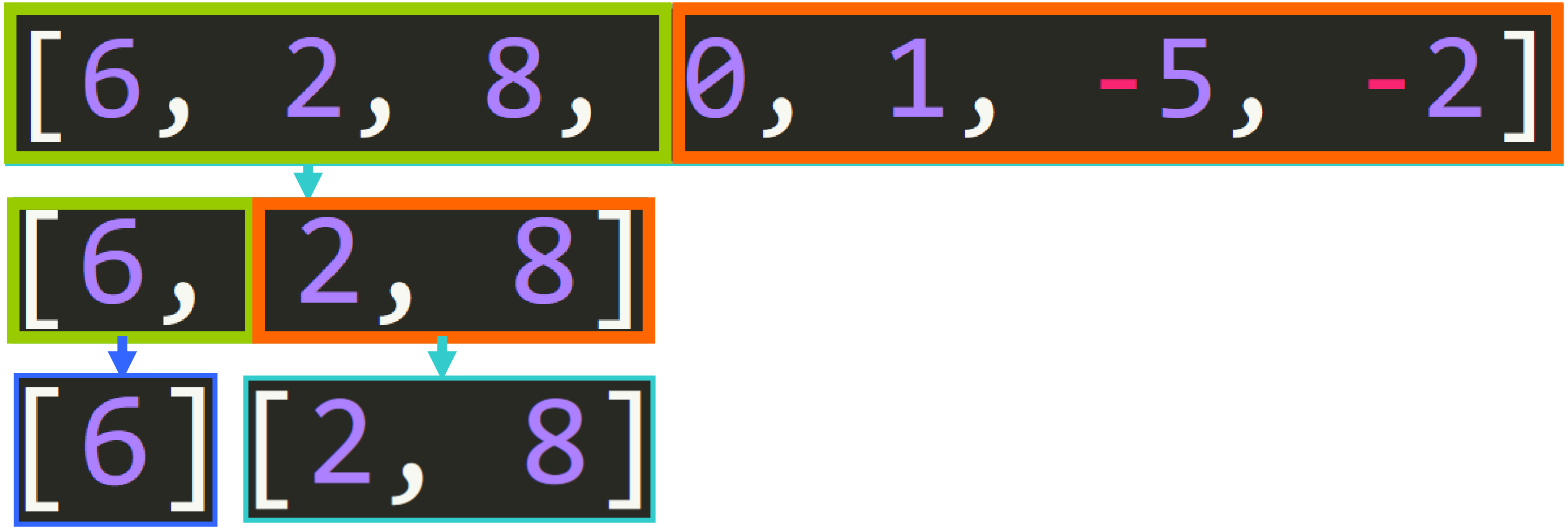
Merge Sort



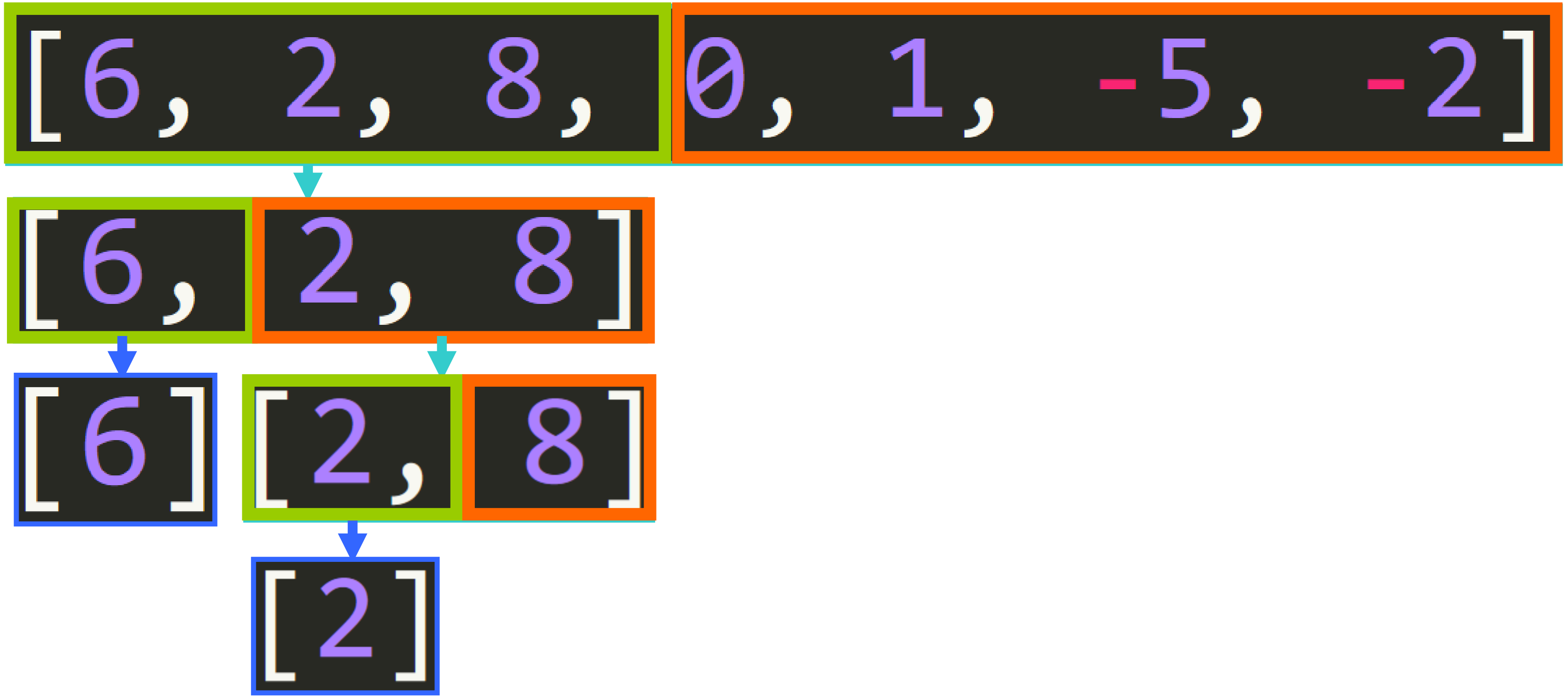
Merge Sort



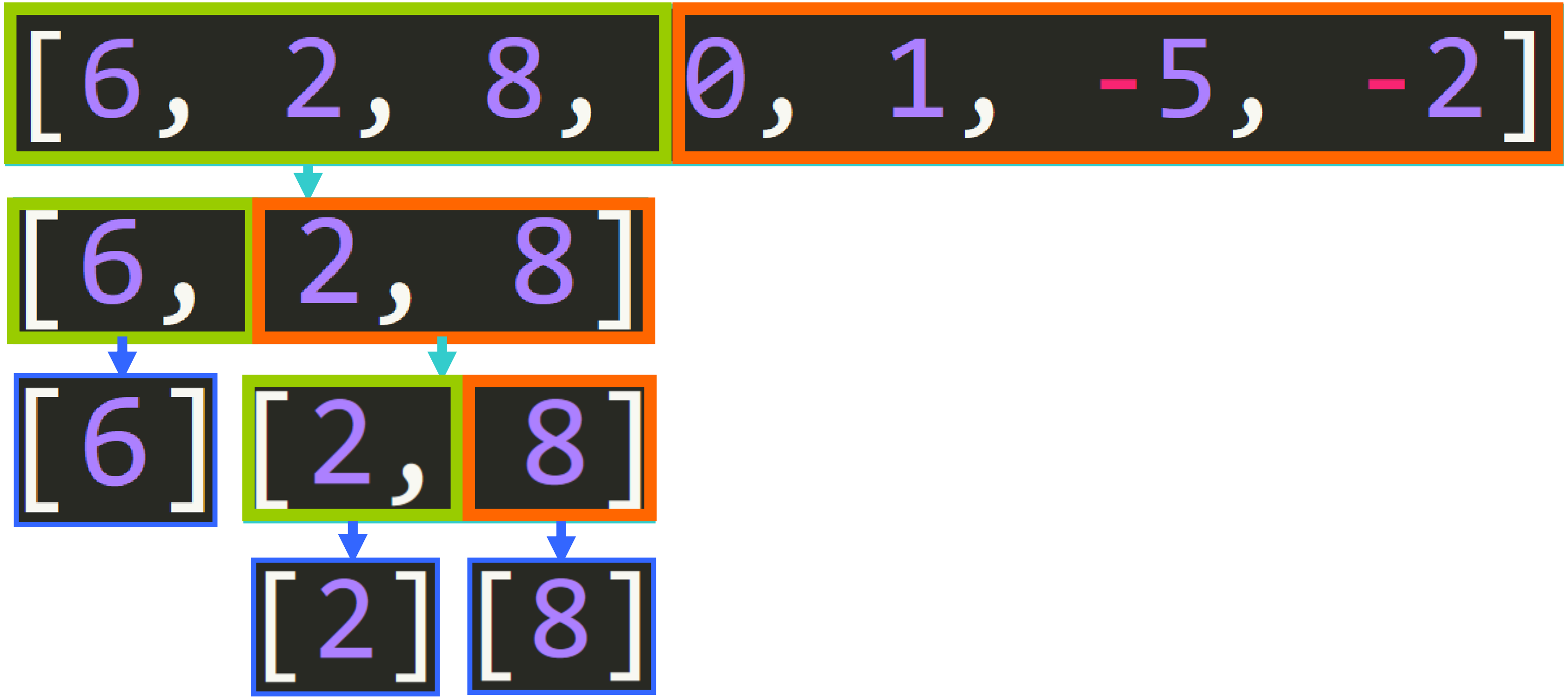
Merge Sort



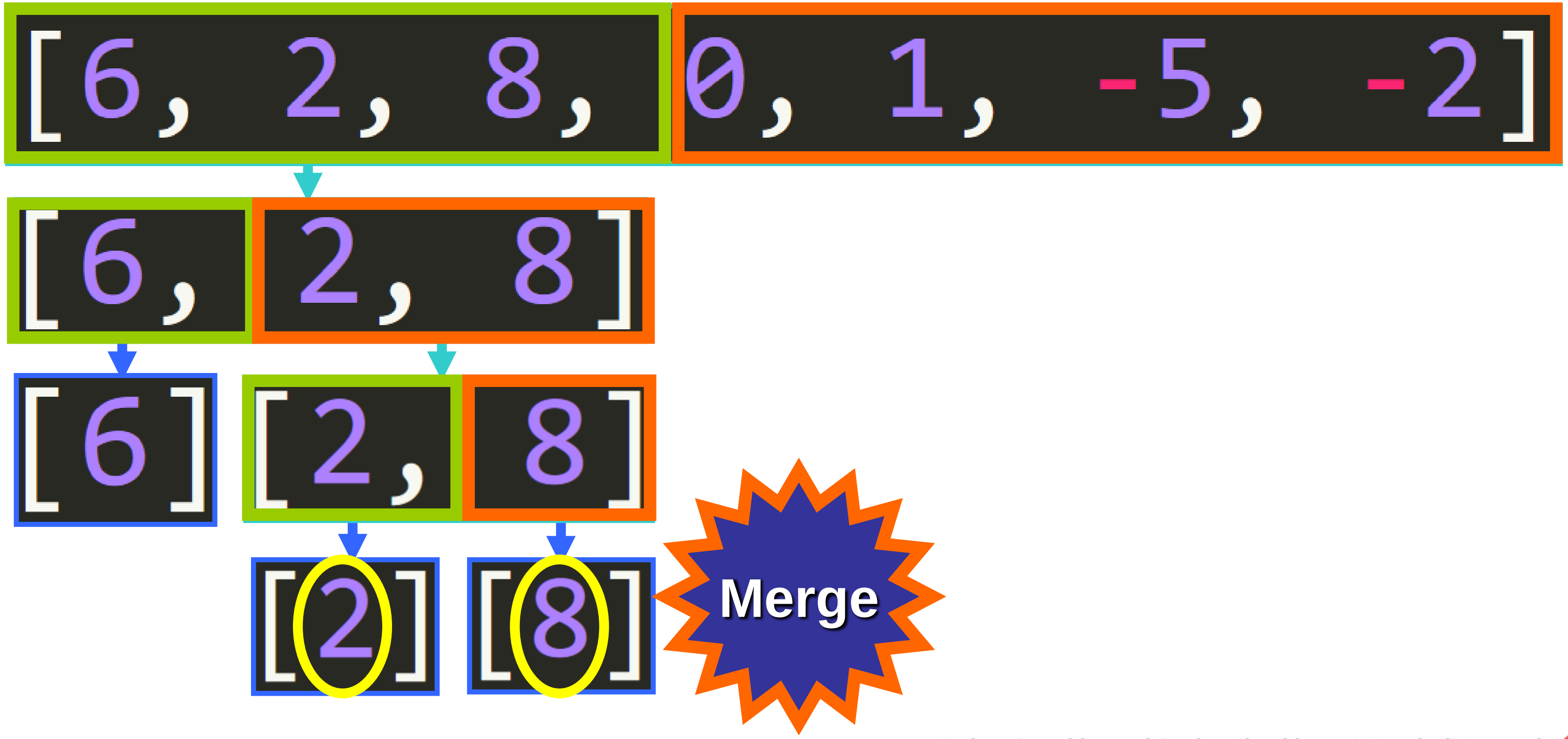
Merge Sort



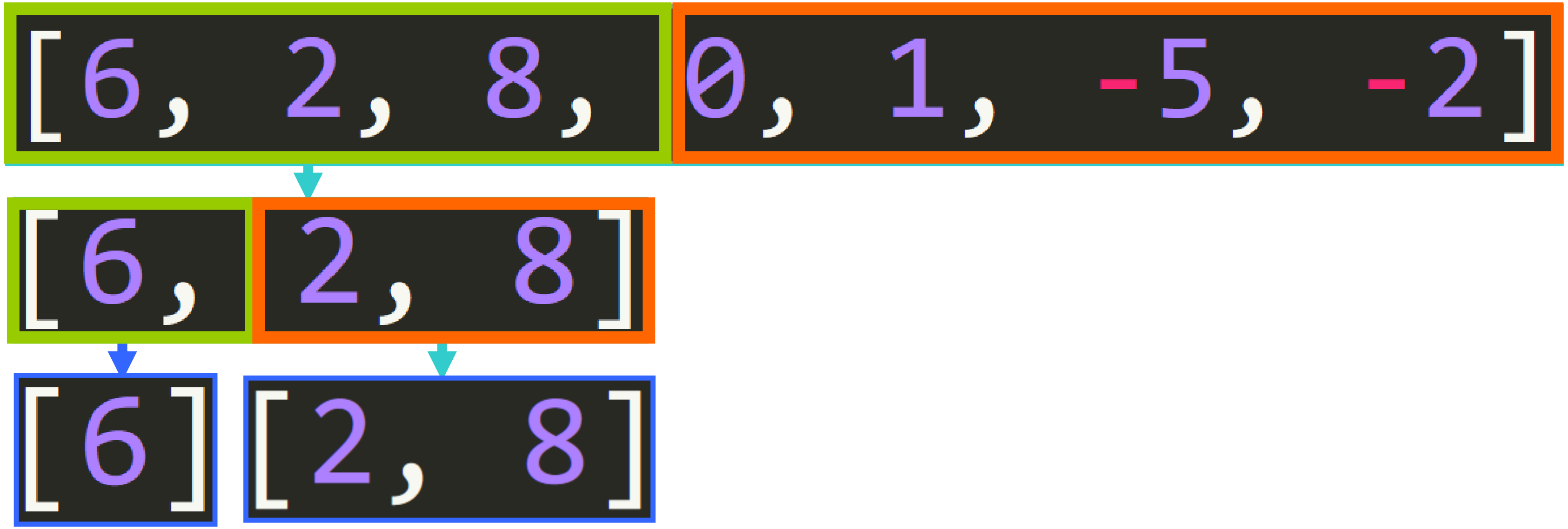
Merge Sort



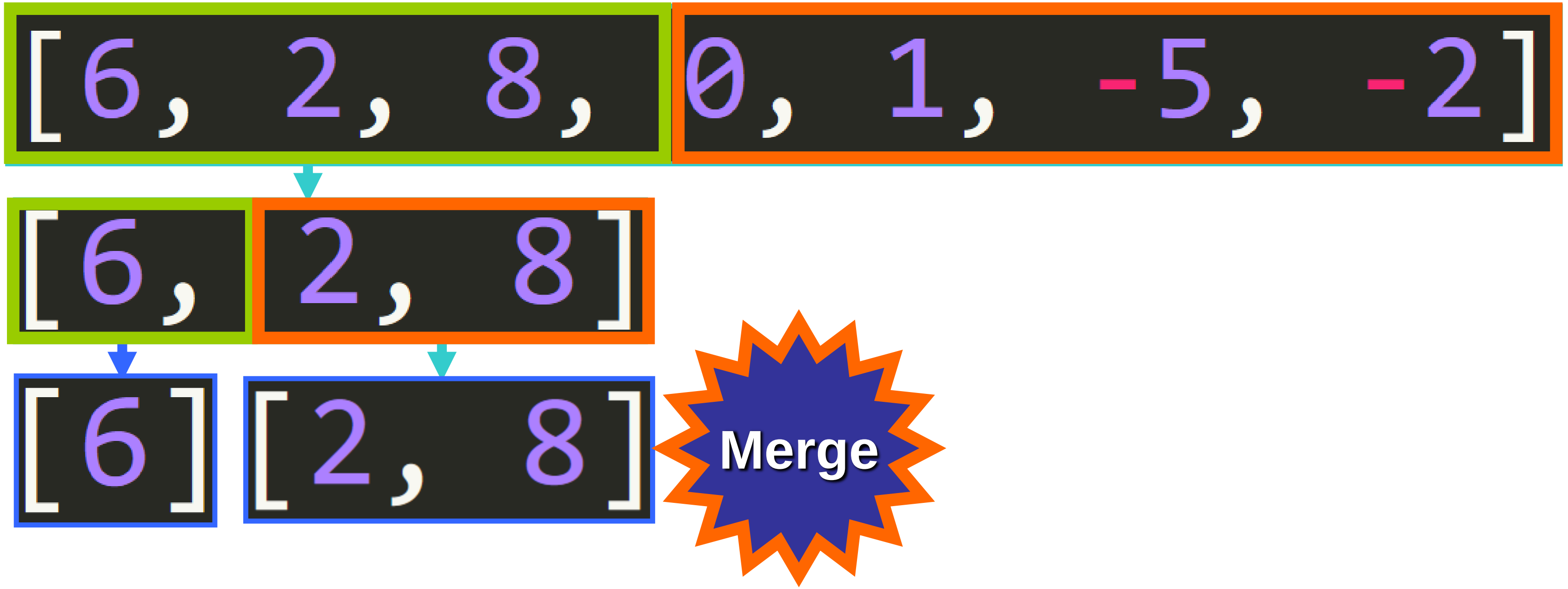
Merge Sort



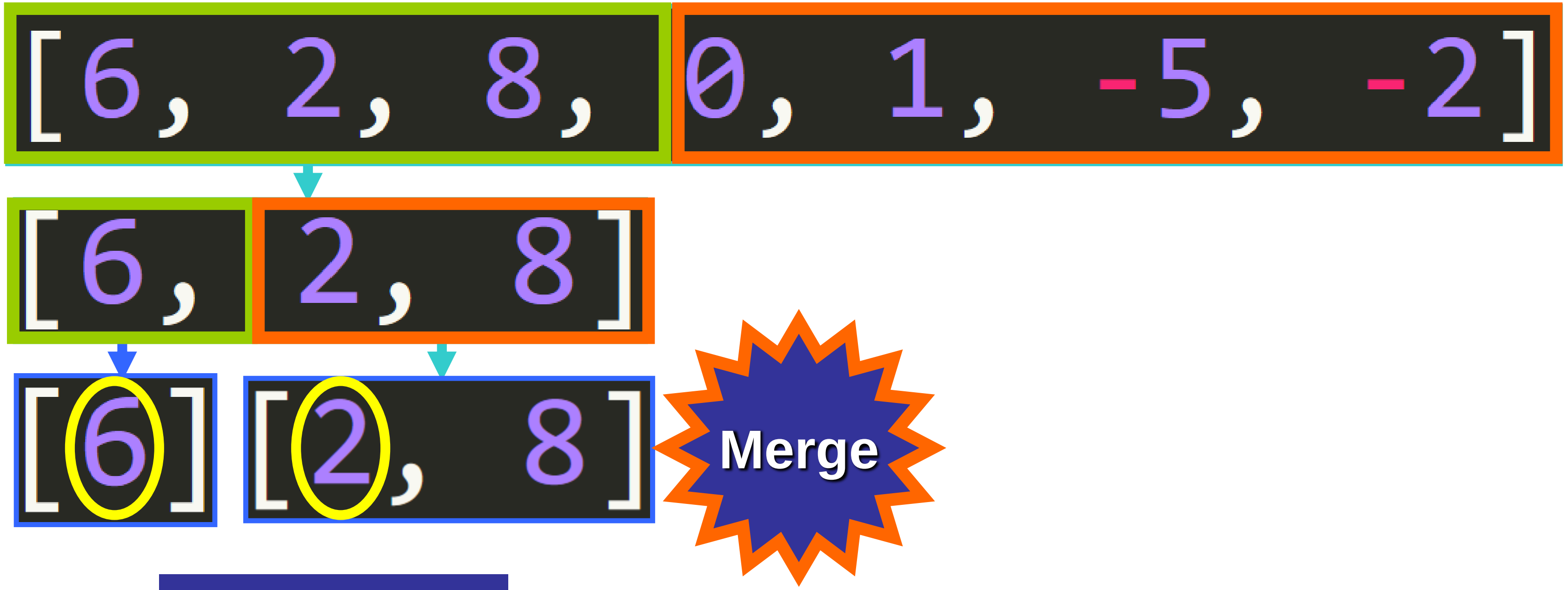
Merge Sort



Merge Sort



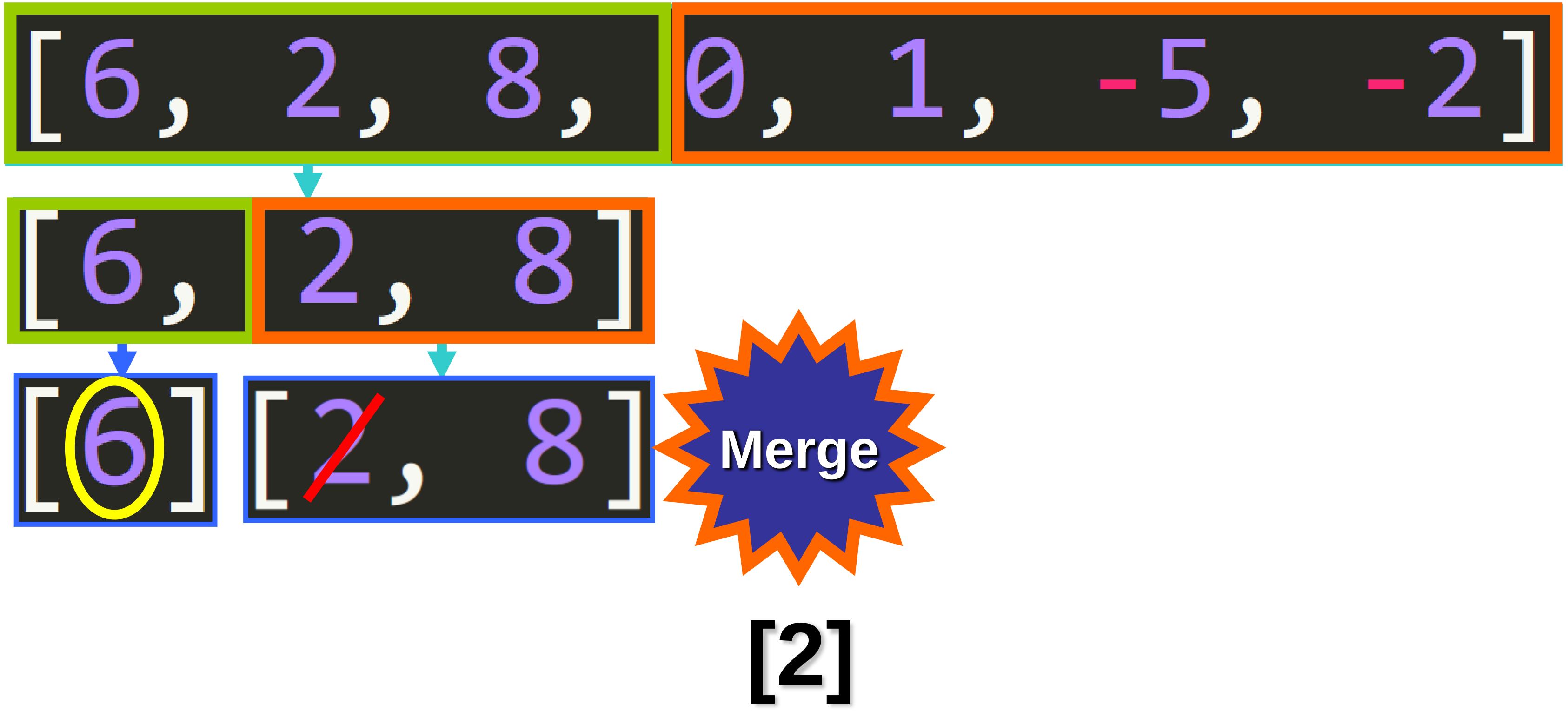
Merge Sort



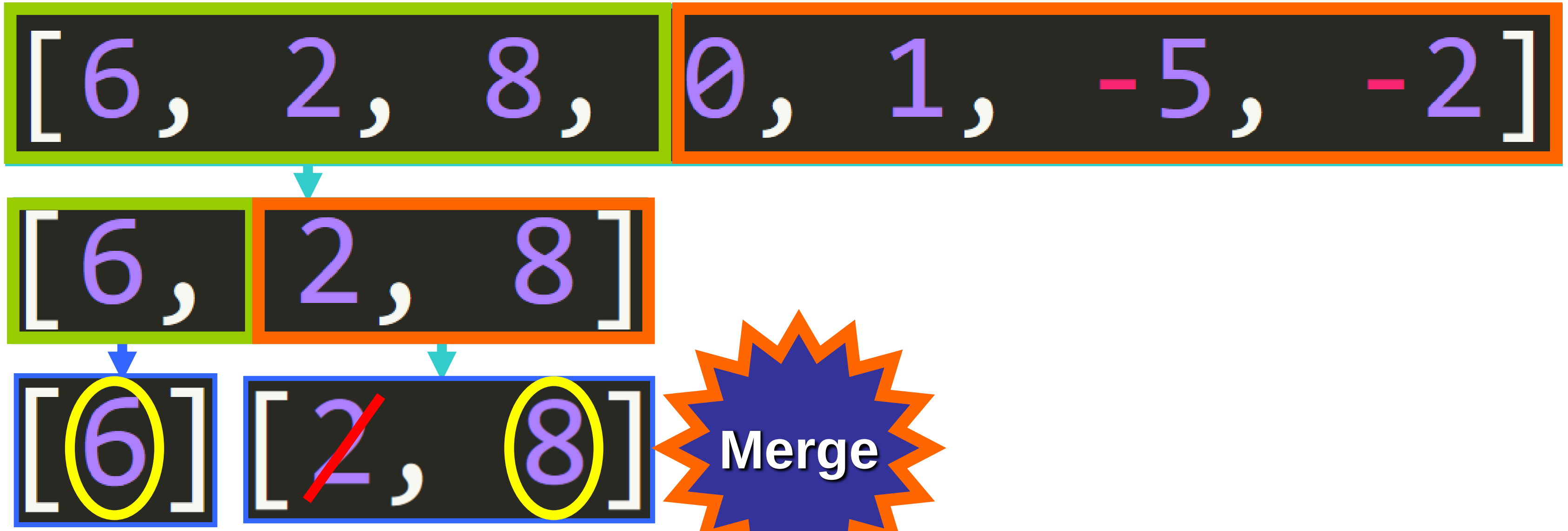
6 < 2?

[ ]

Merge Sort



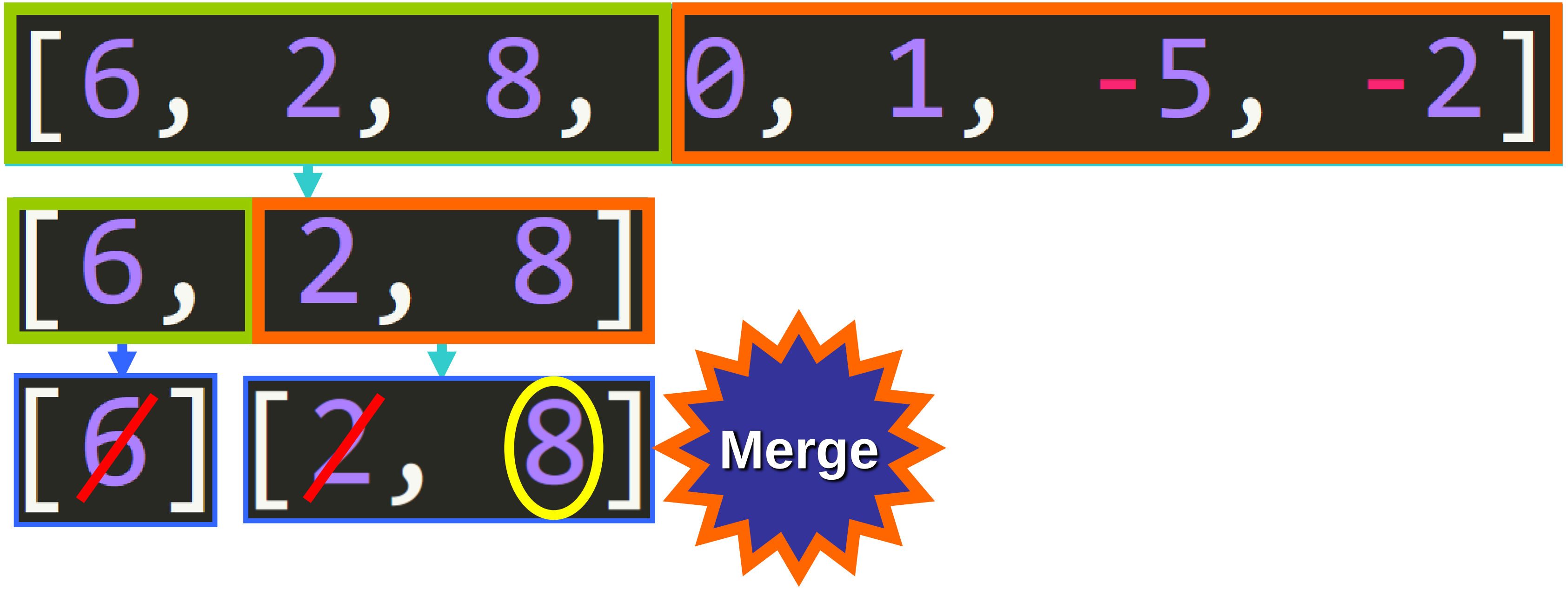
Merge Sort



6 < 8?

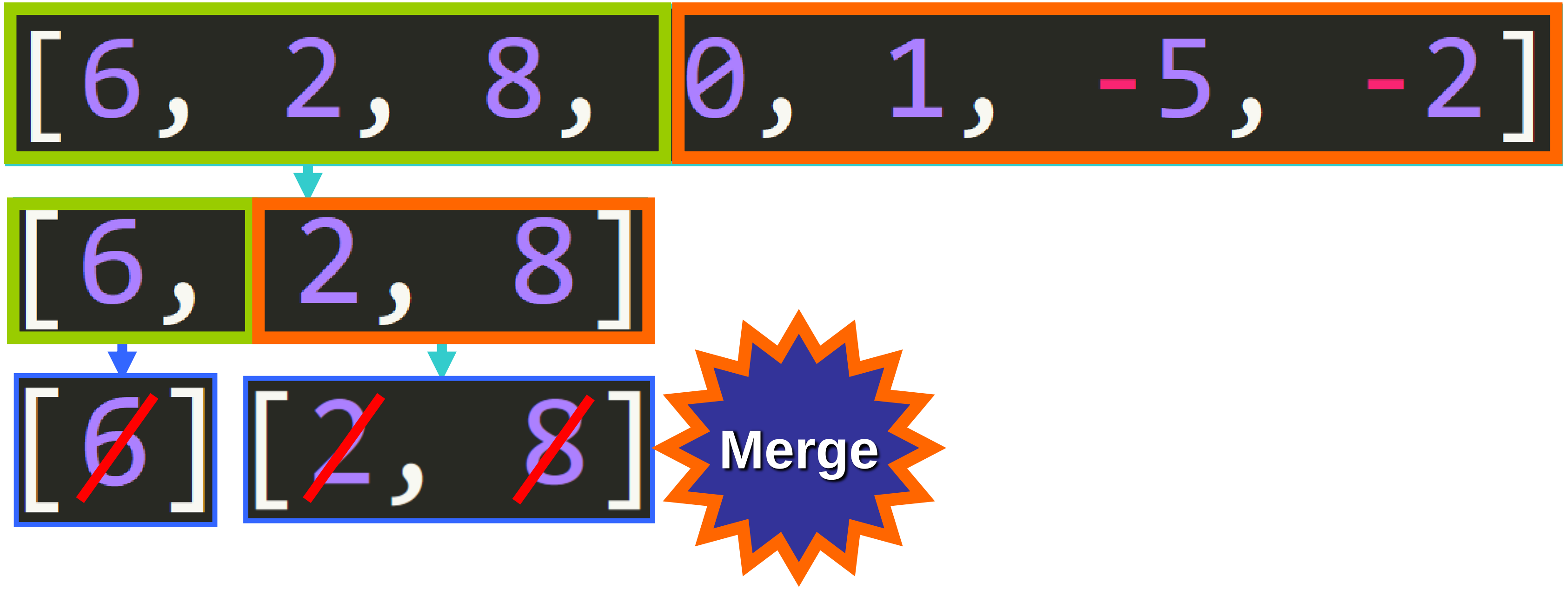
[2]

Merge Sort



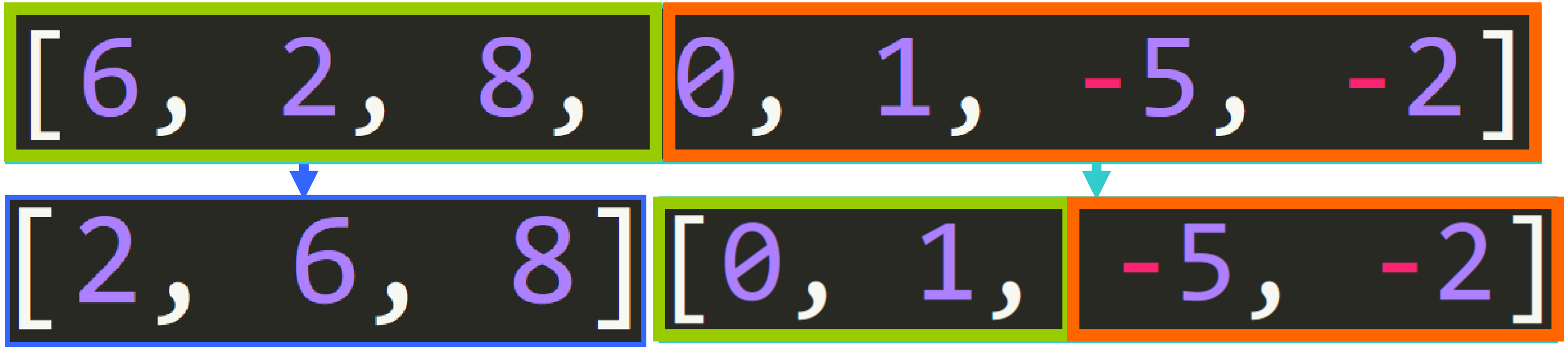
[2, 6]

Merge Sort



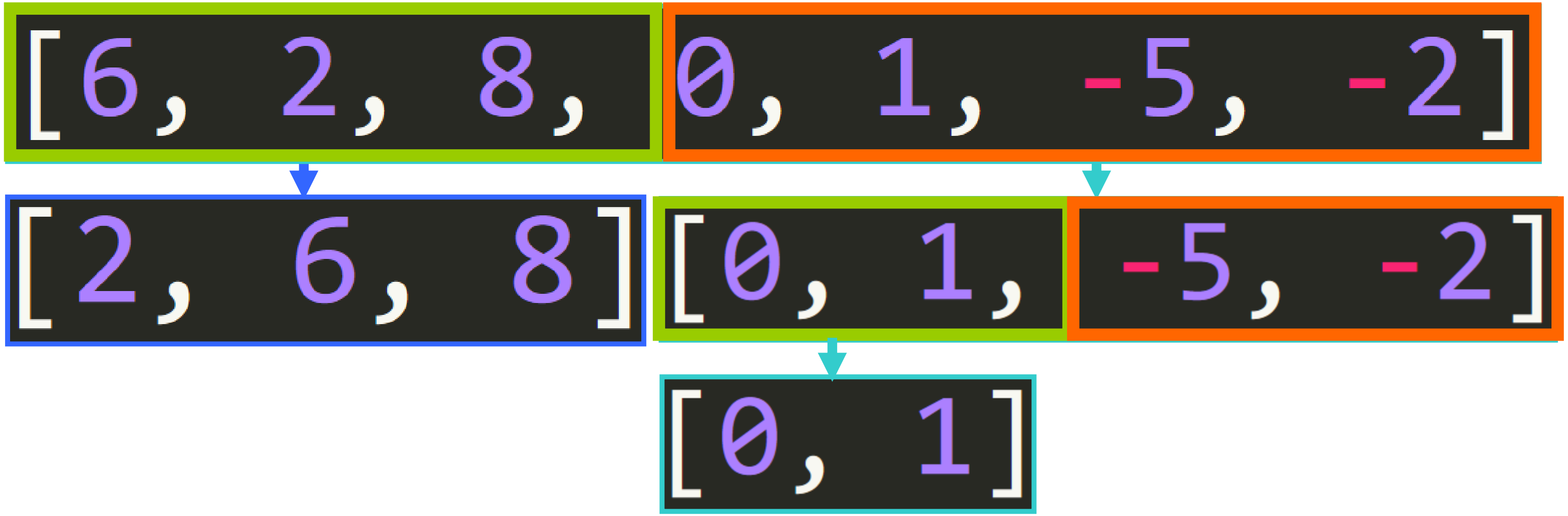
[2, 6, 8]

Merge Sort



```
[6, 2, 8, 0, 1, -5, -2]
[2, 6, 8][0, 1, -5, -2]
```

Merge Sort



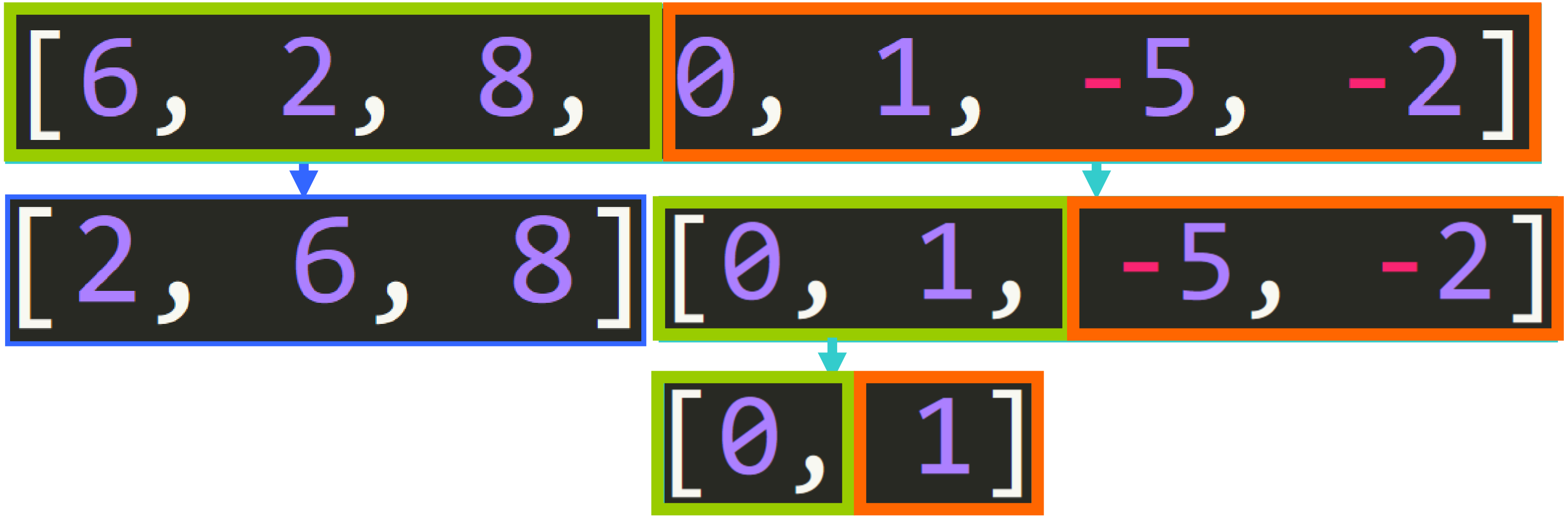
```
[6, 2, 8, 0, 1, -5, -2]
```

```
[2, 6, 8]
```

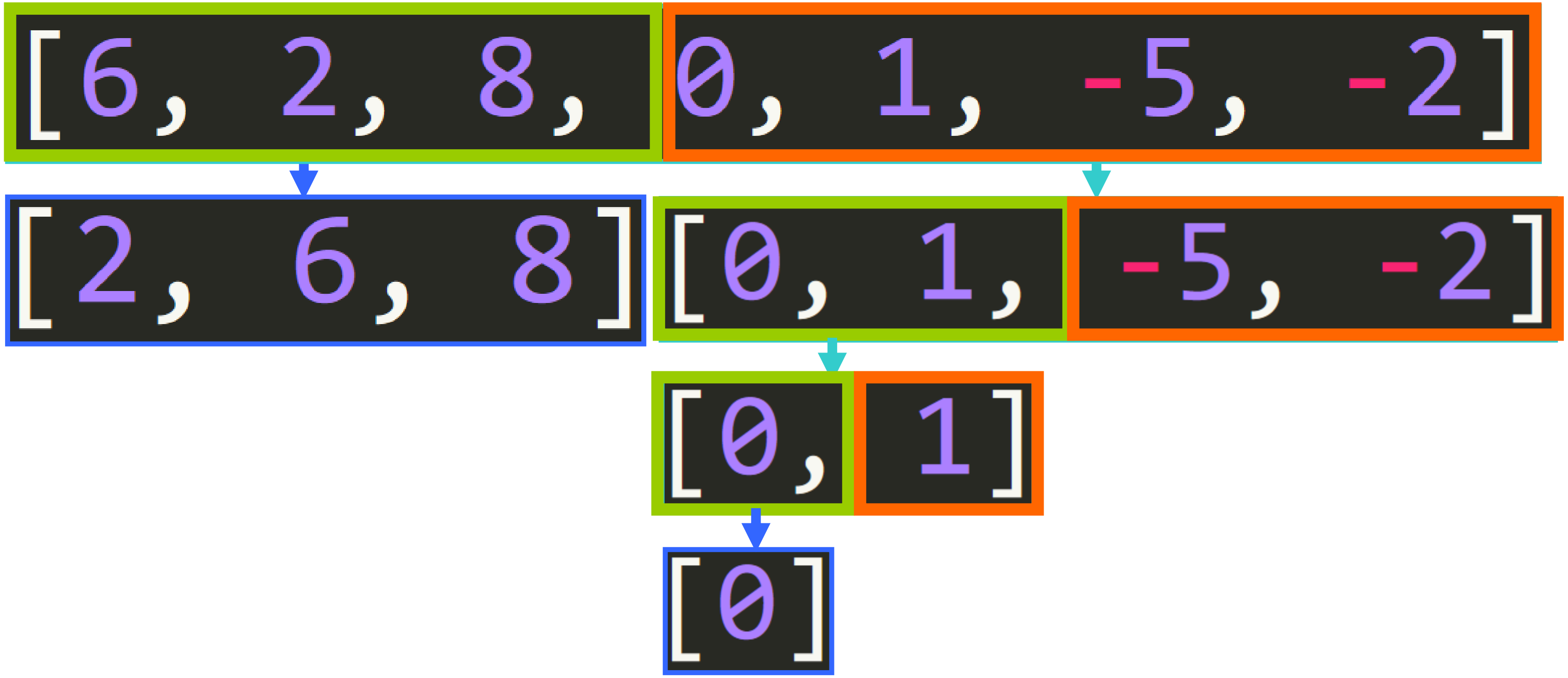
```
[0, 1, -5, -2]
```

```
[0, 1]
```

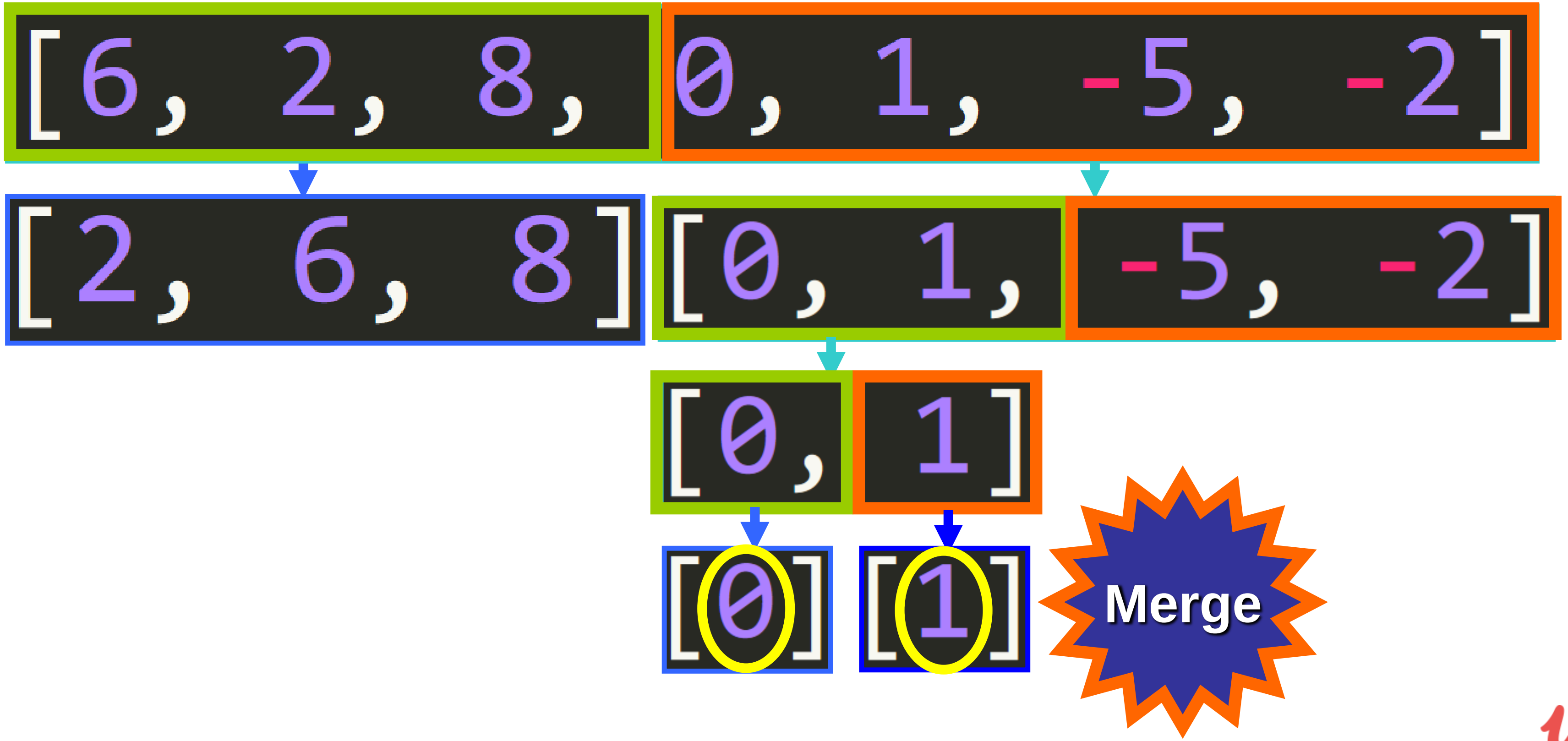
Merge Sort



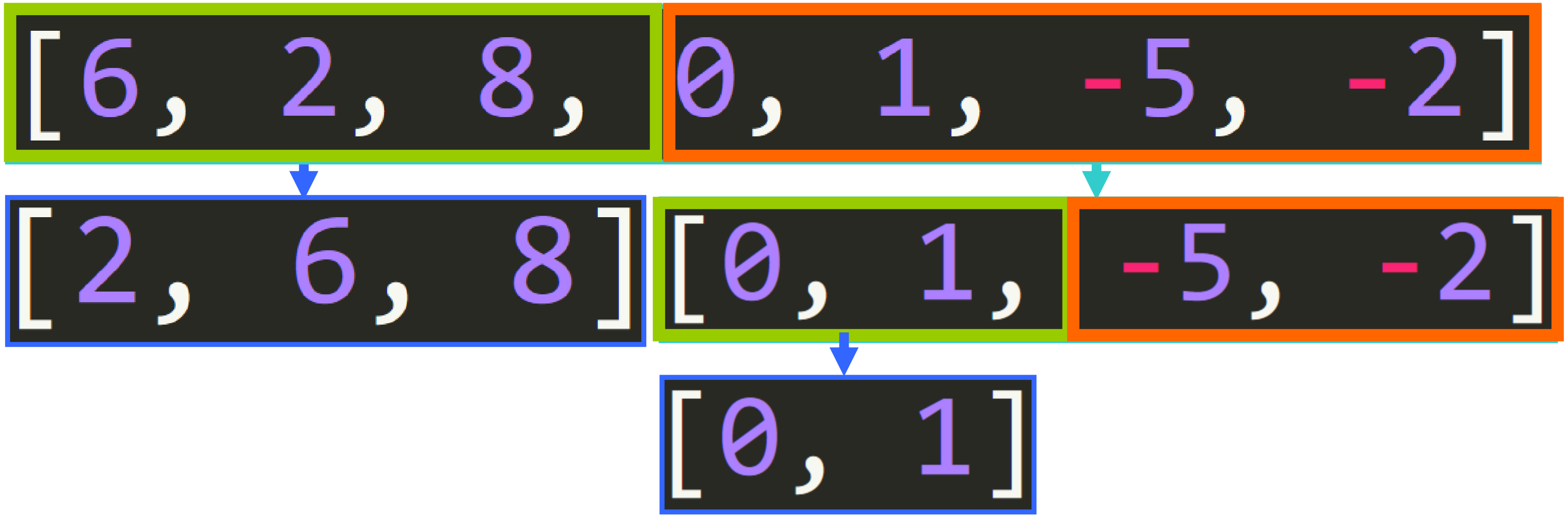
Merge Sort



Merge Sort

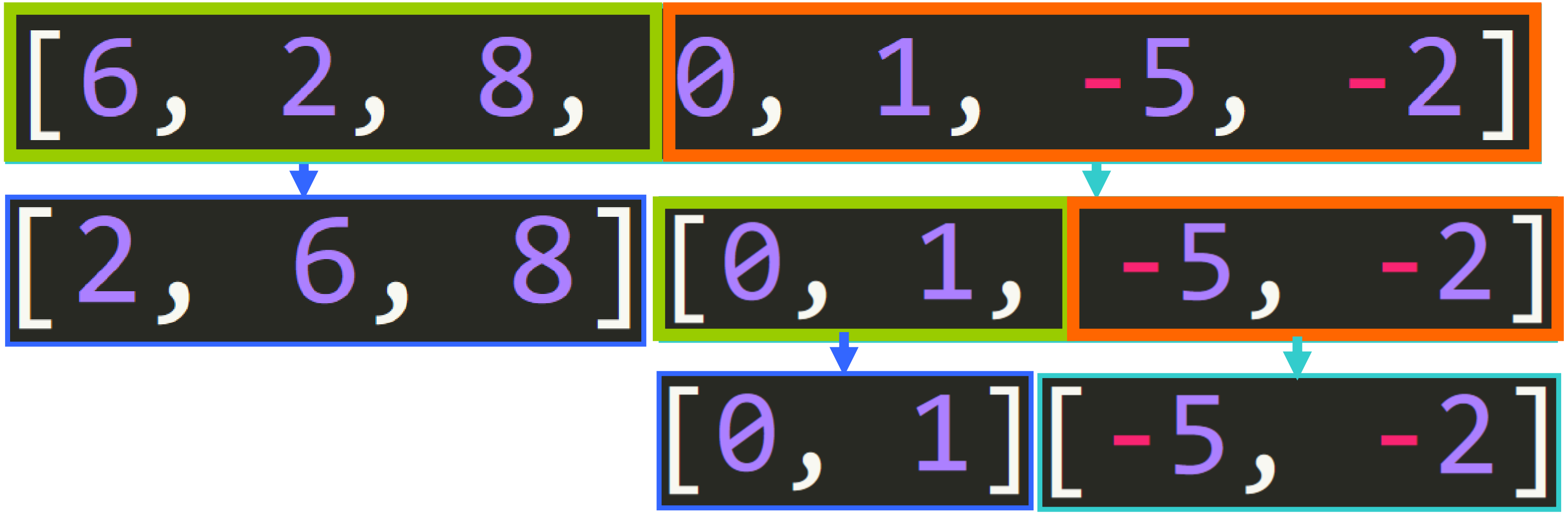


Merge Sort

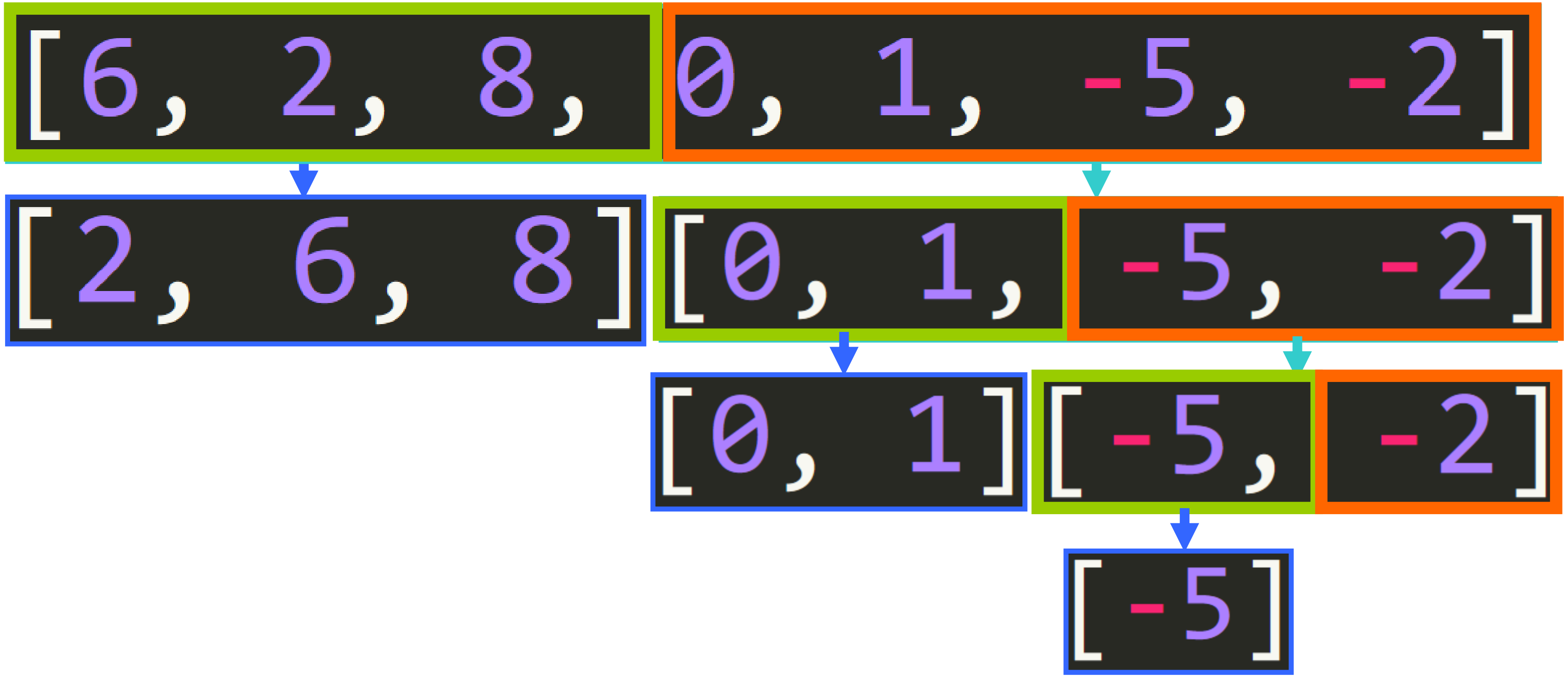


```
[6, 2, 8, 0, 1, -5, -2]
[2, 6, 8]  [0, 1, -5, -2]
           [0, 1]
```

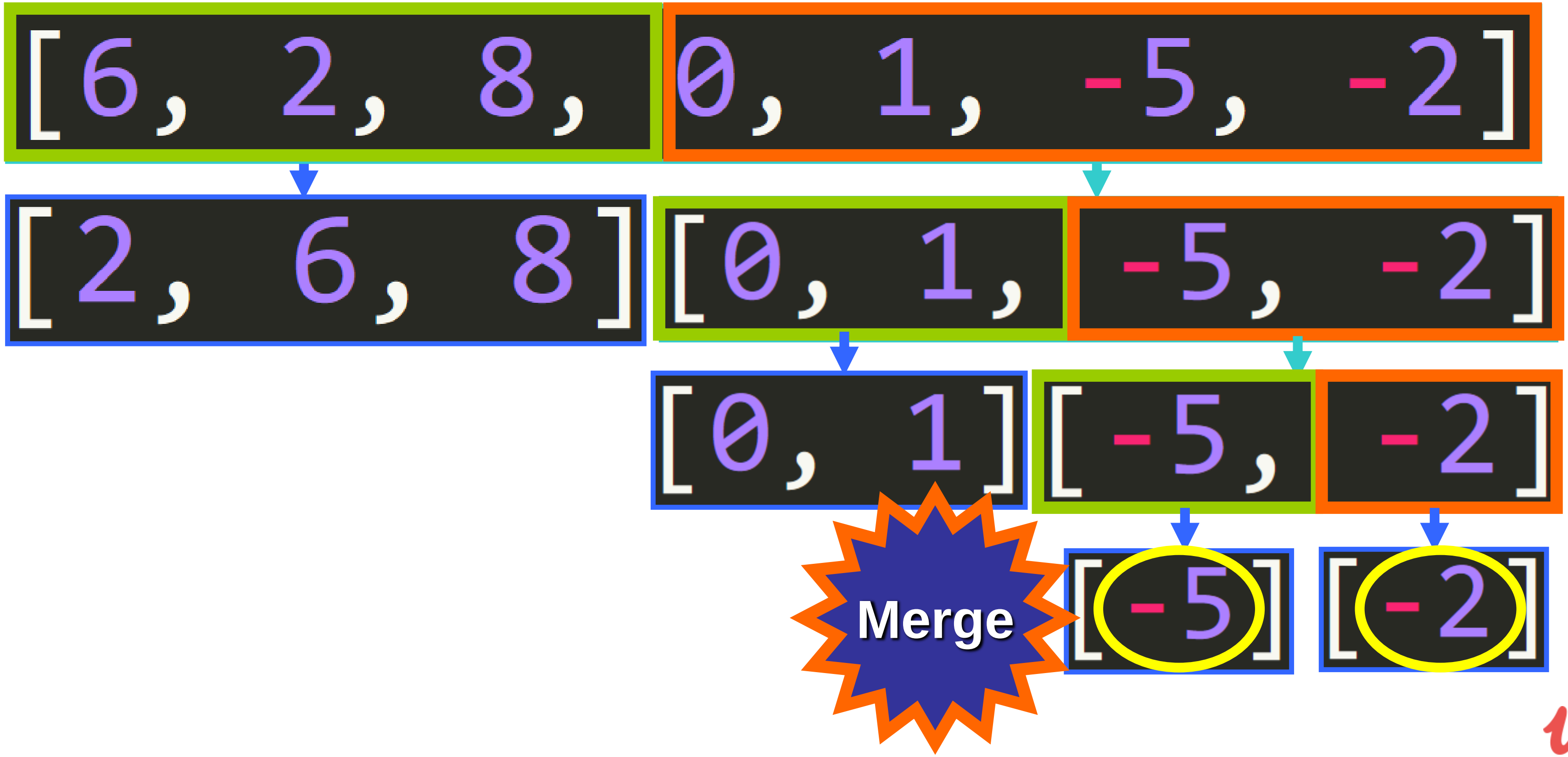
Merge Sort



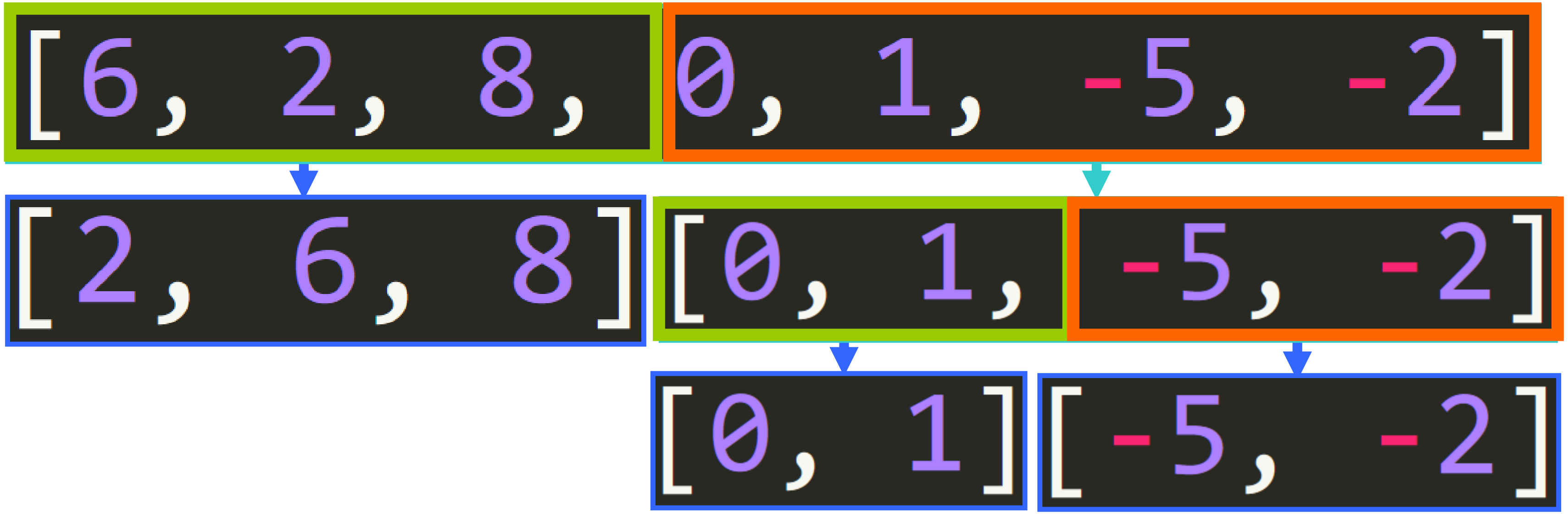
Merge Sort



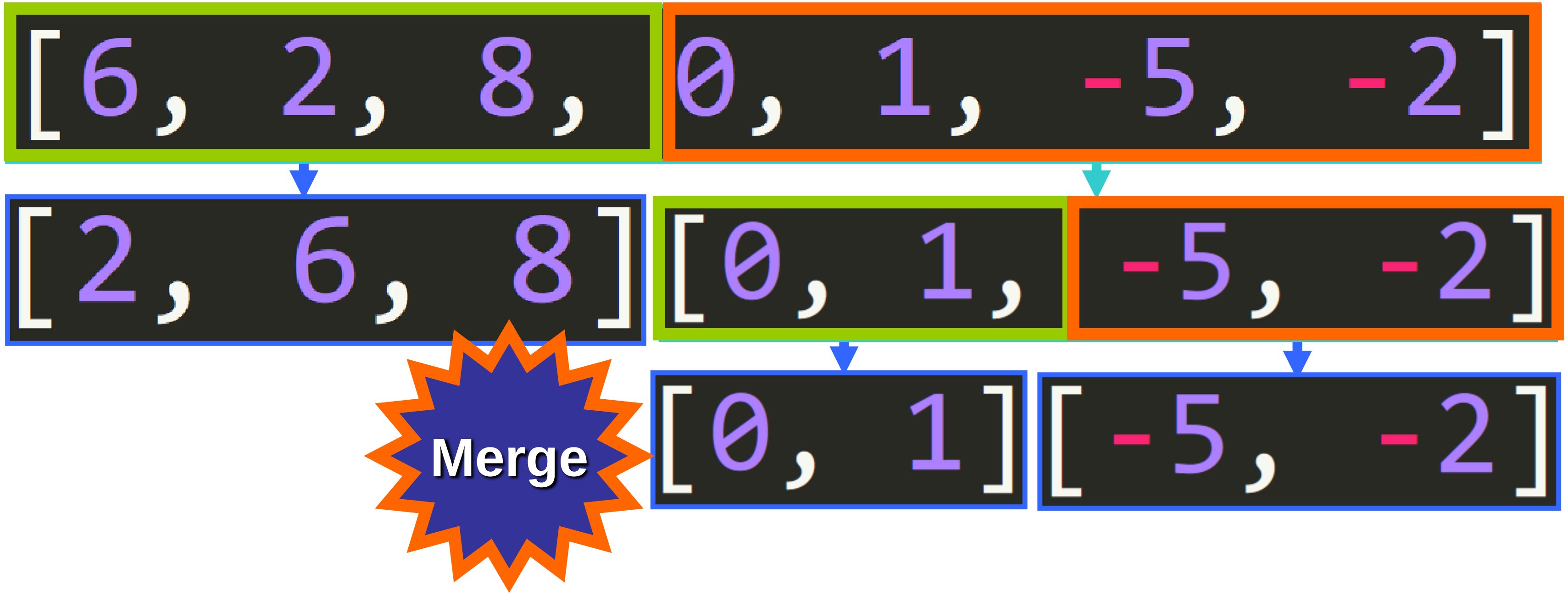
Merge Sort



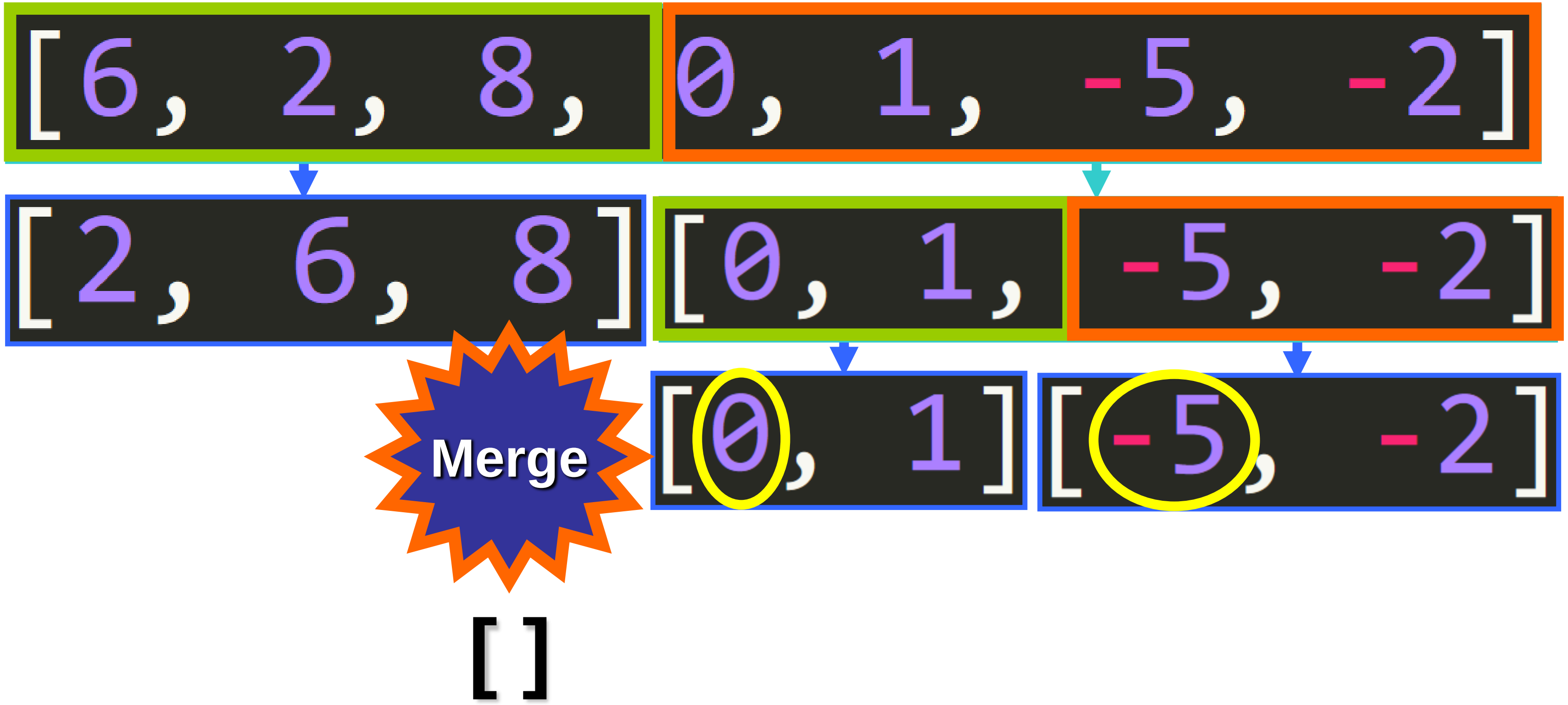
Merge Sort



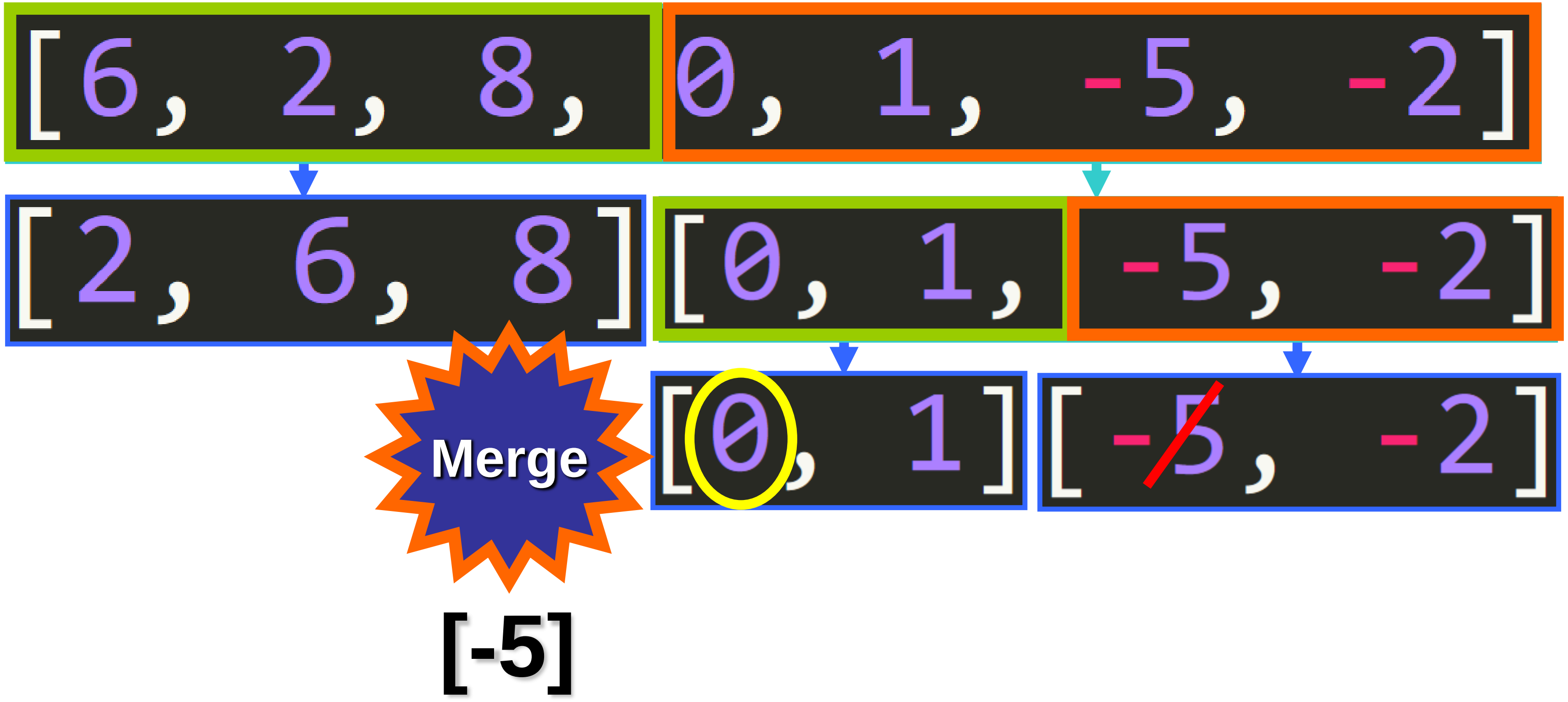
Merge Sort



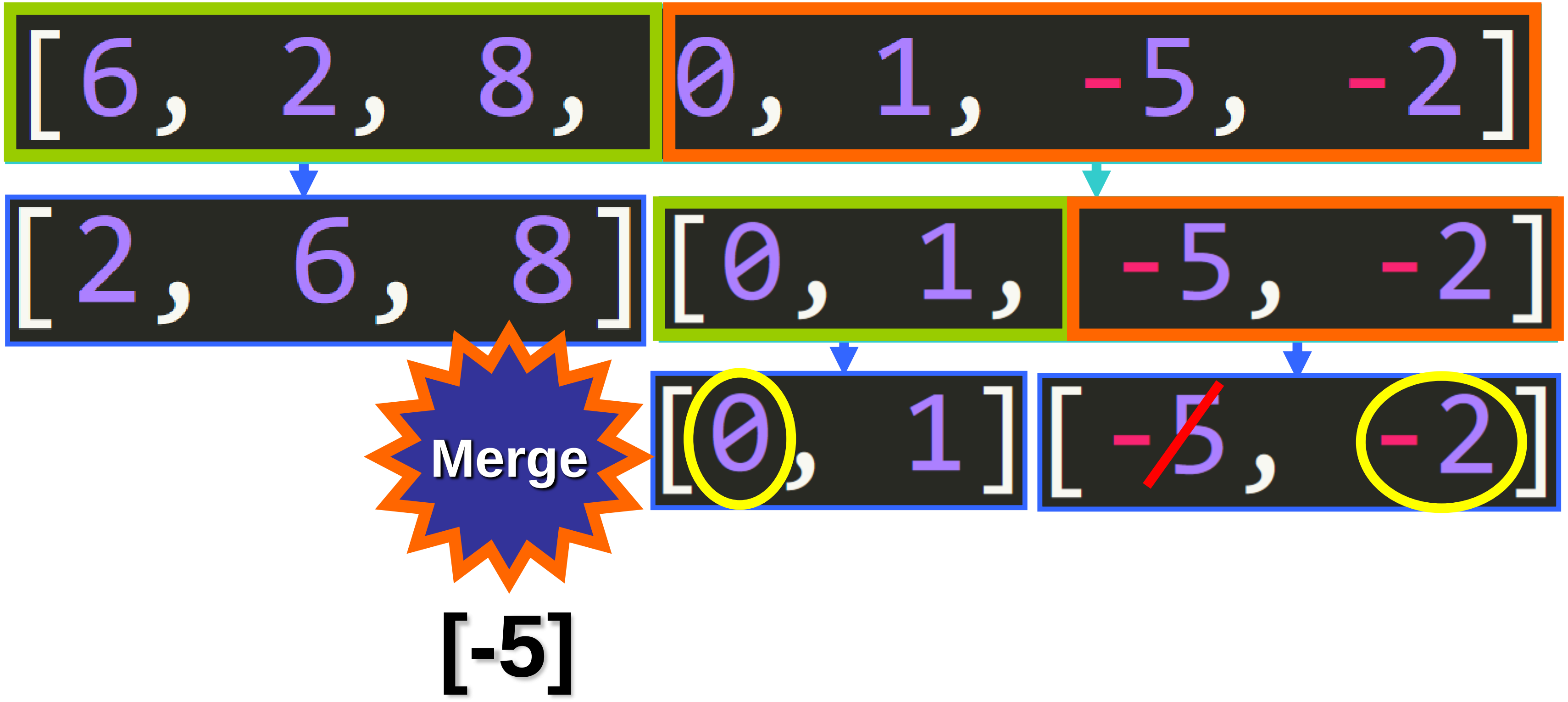
Merge Sort



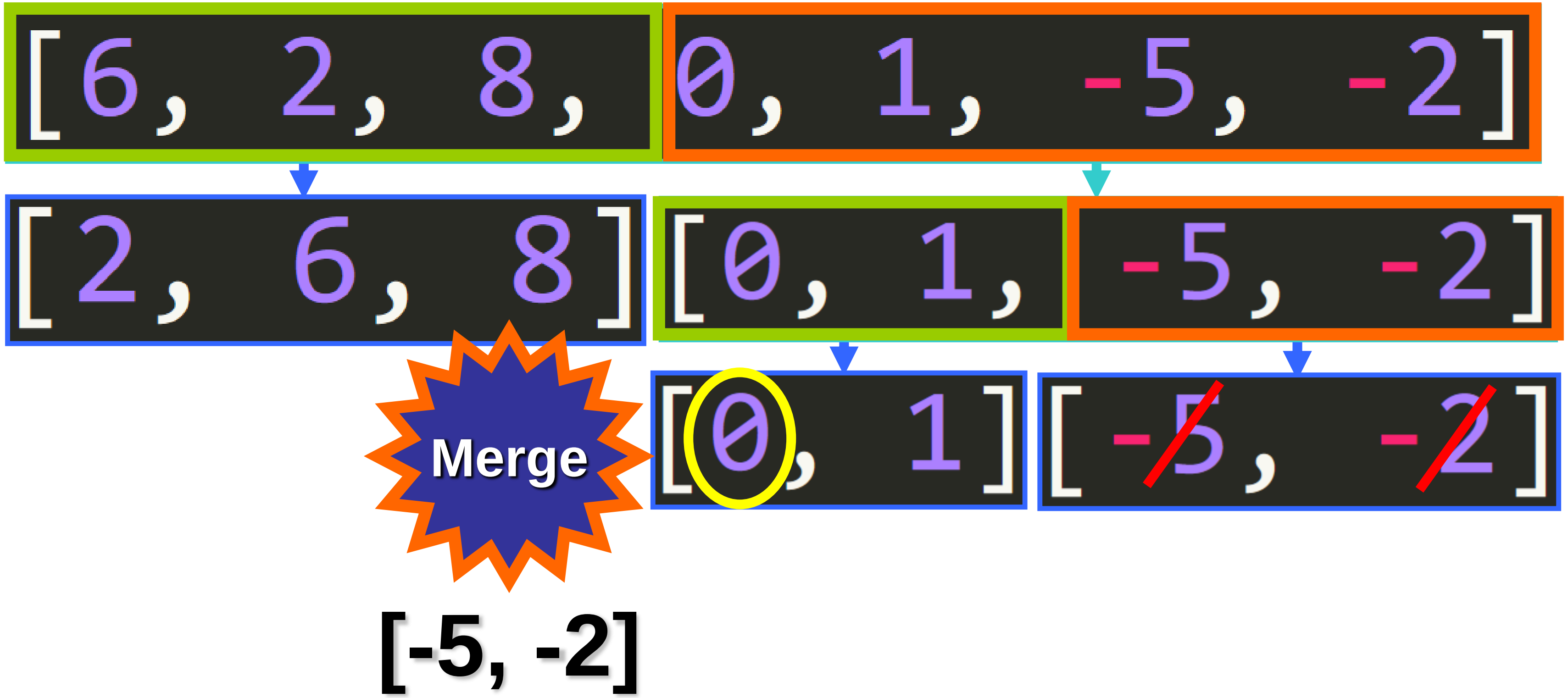
Merge Sort



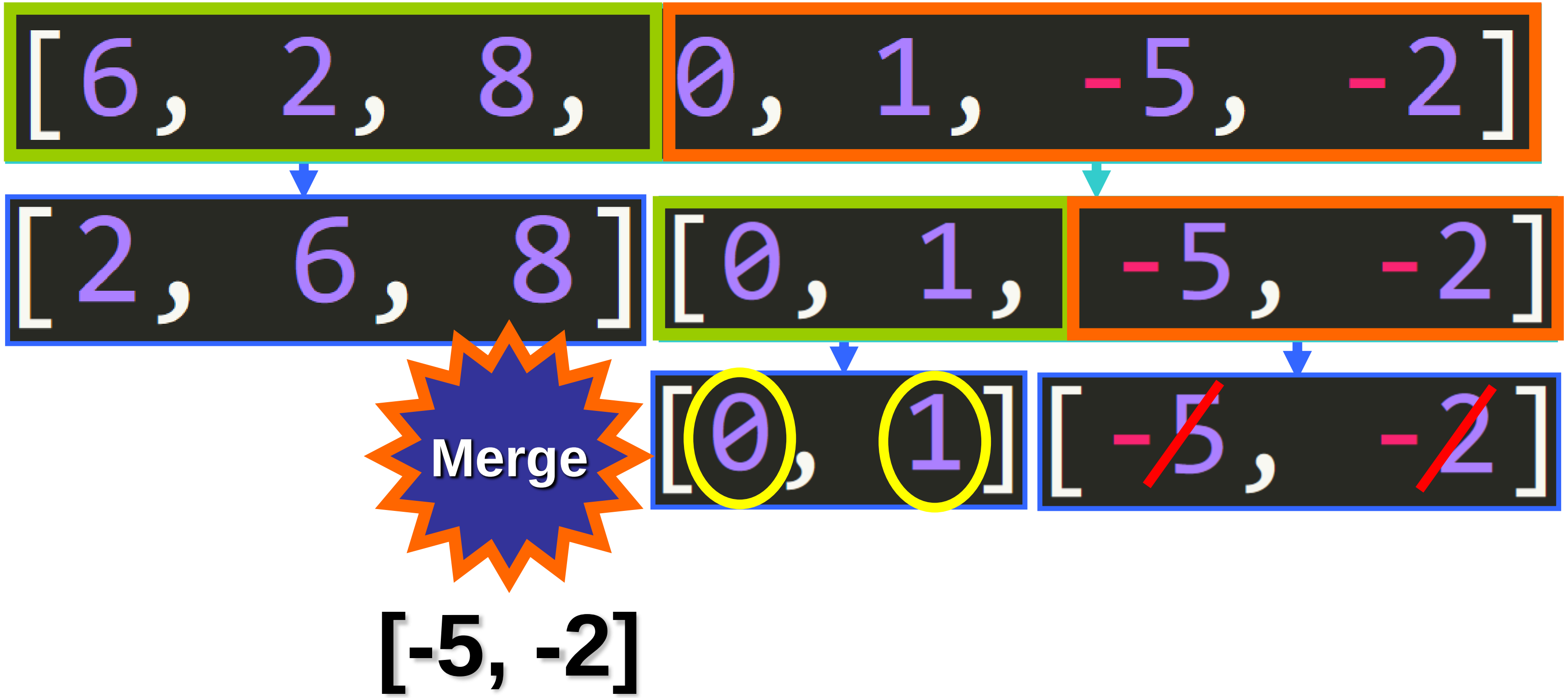
Merge Sort



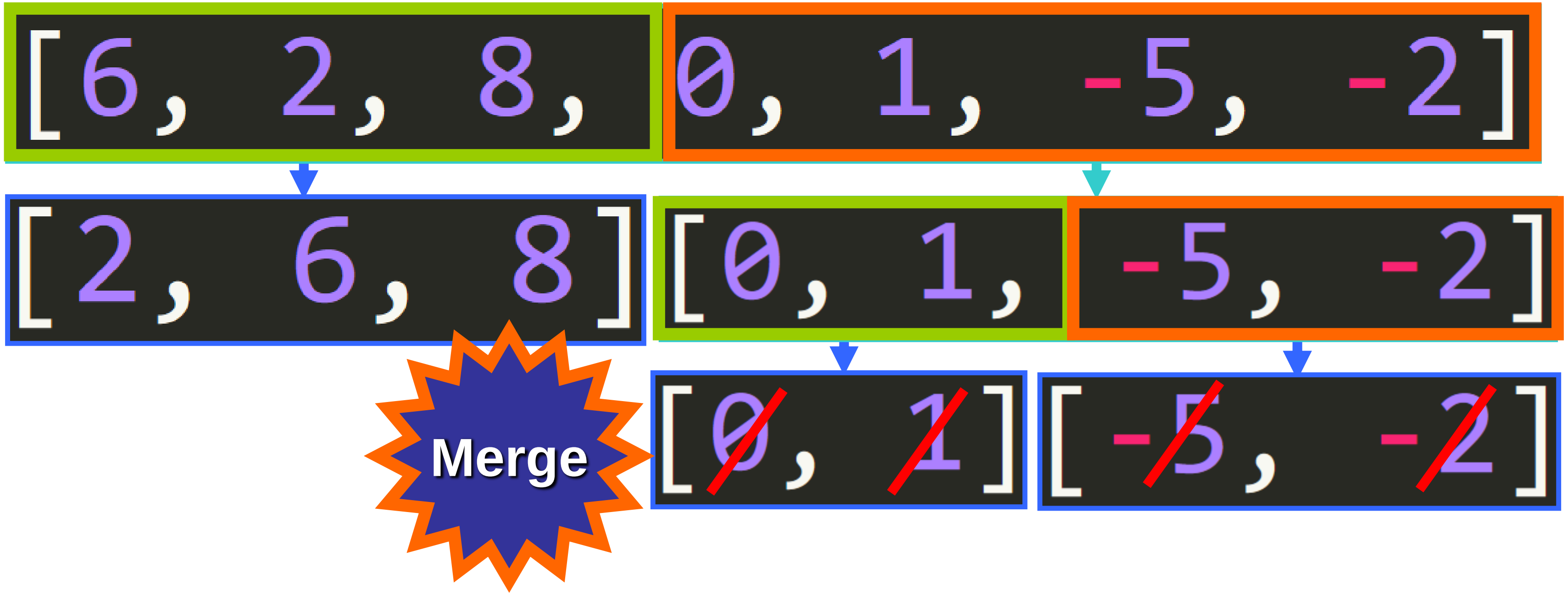
Merge Sort



Merge Sort

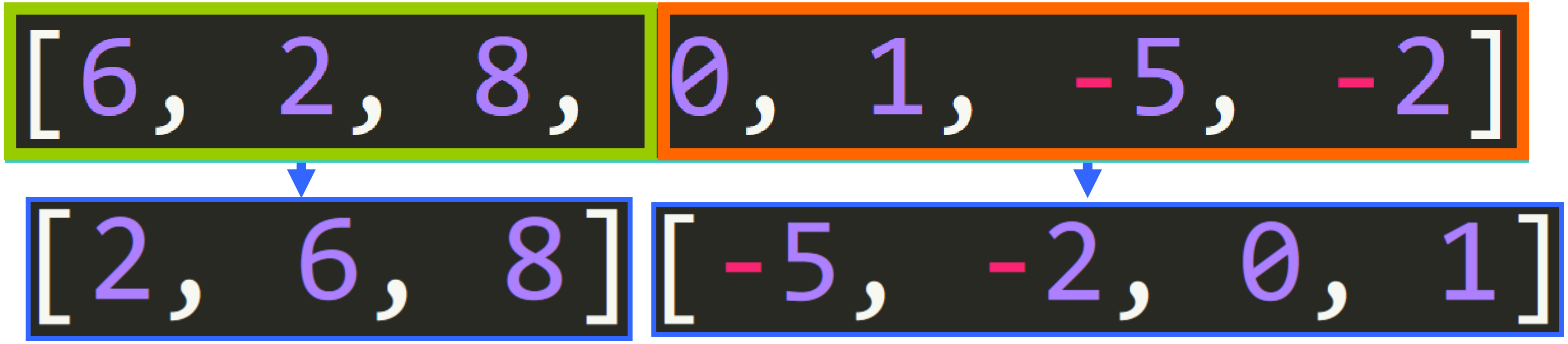


Merge Sort

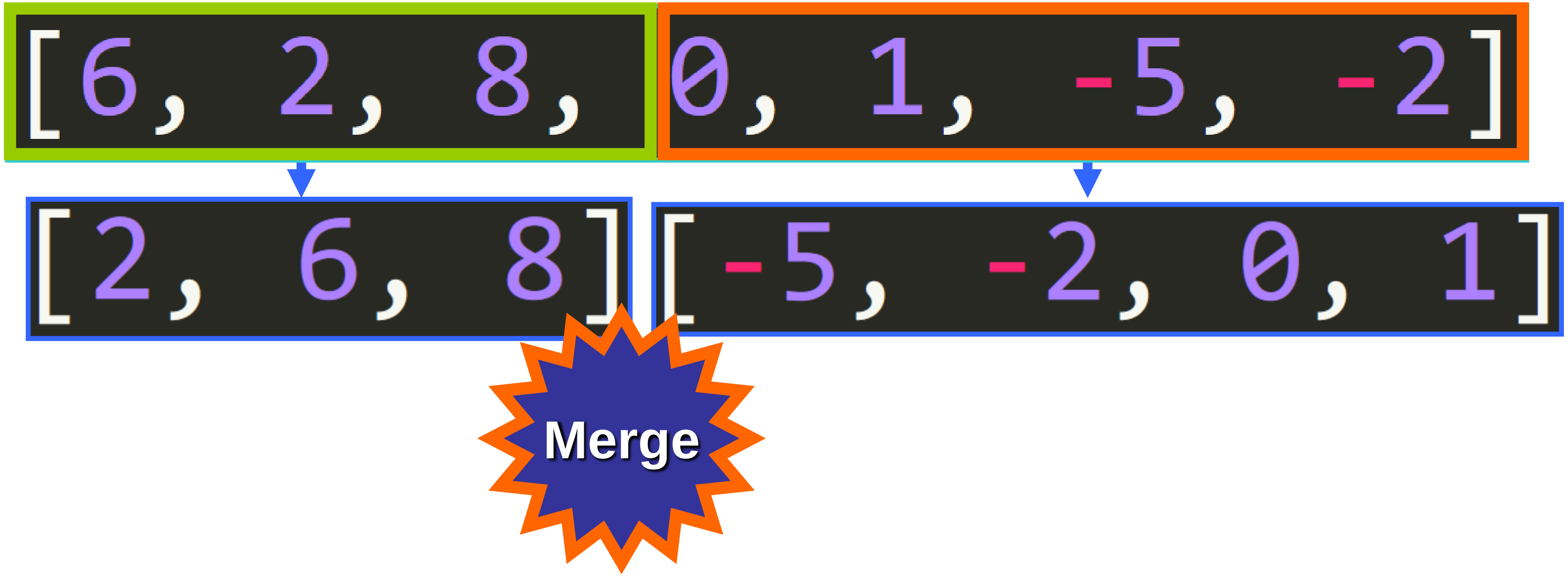


[-5, -2, 0, 1]

Merge Sort



Merge Sort

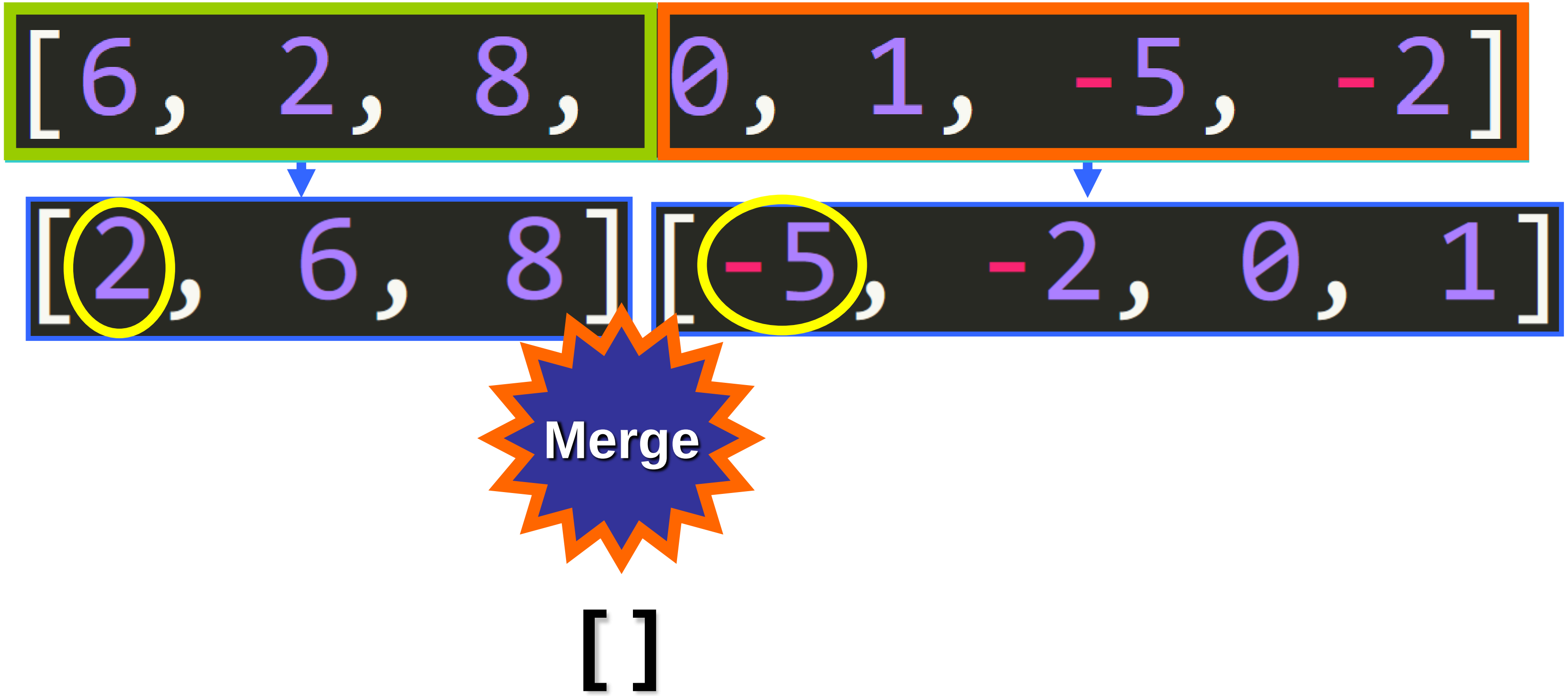


```
[6, 2, 8, 0, 1, -5, -2]
```

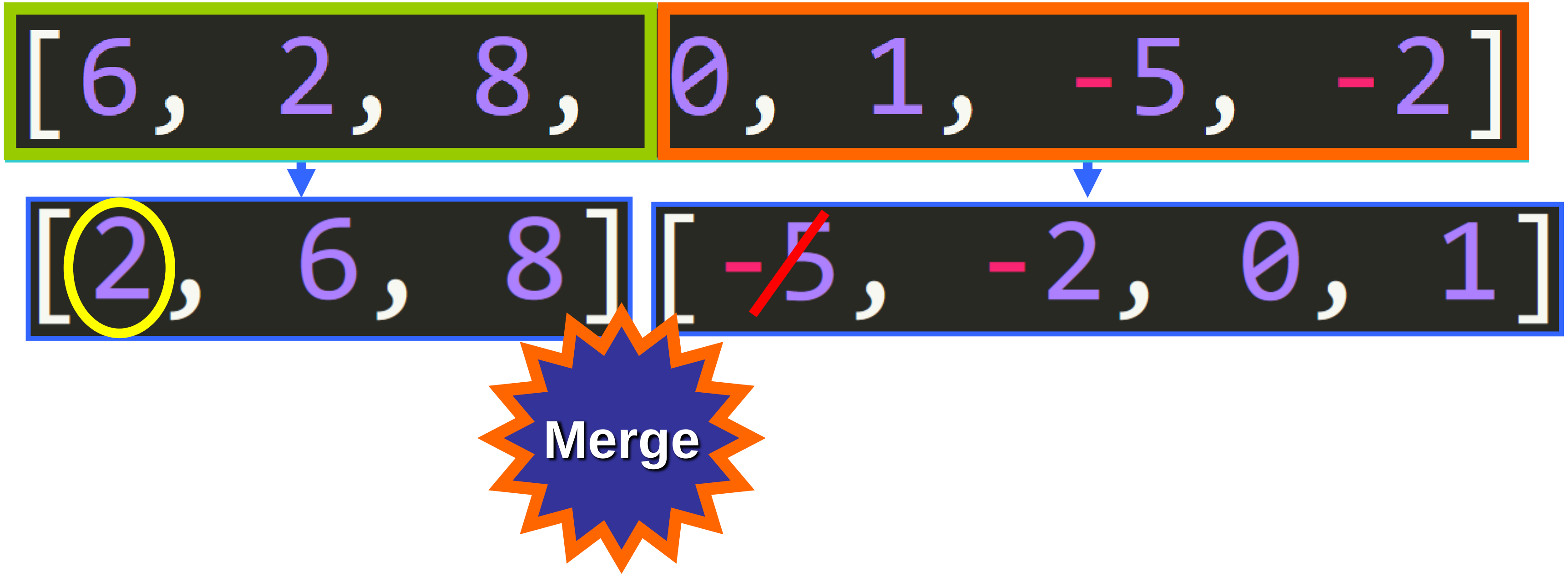
```
[2, 6, 8]
```

```
[-5, -2, 0, 1]
```

Merge Sort



Merge Sort



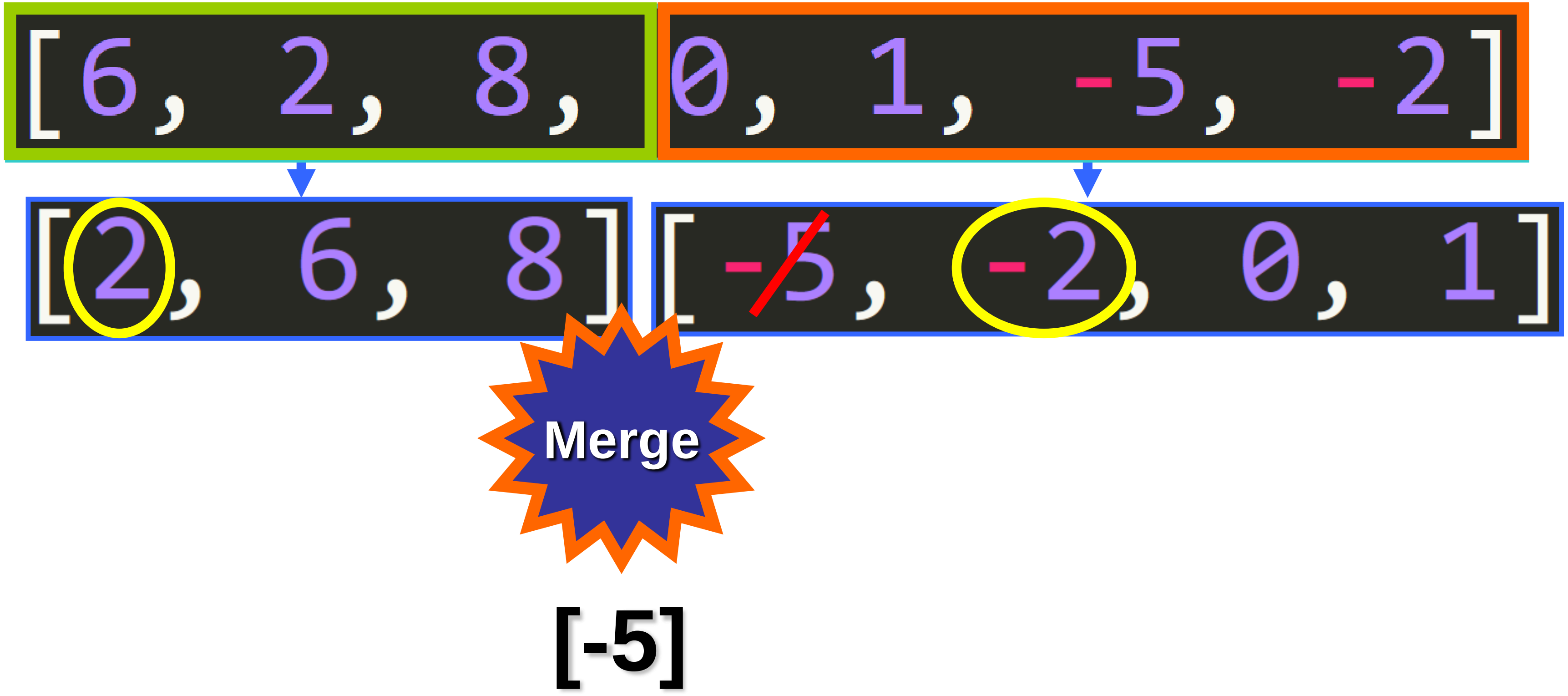
```
[6, 2, 8, 0, 1, -5, -2]
```

```
[2, 6, 8]
```

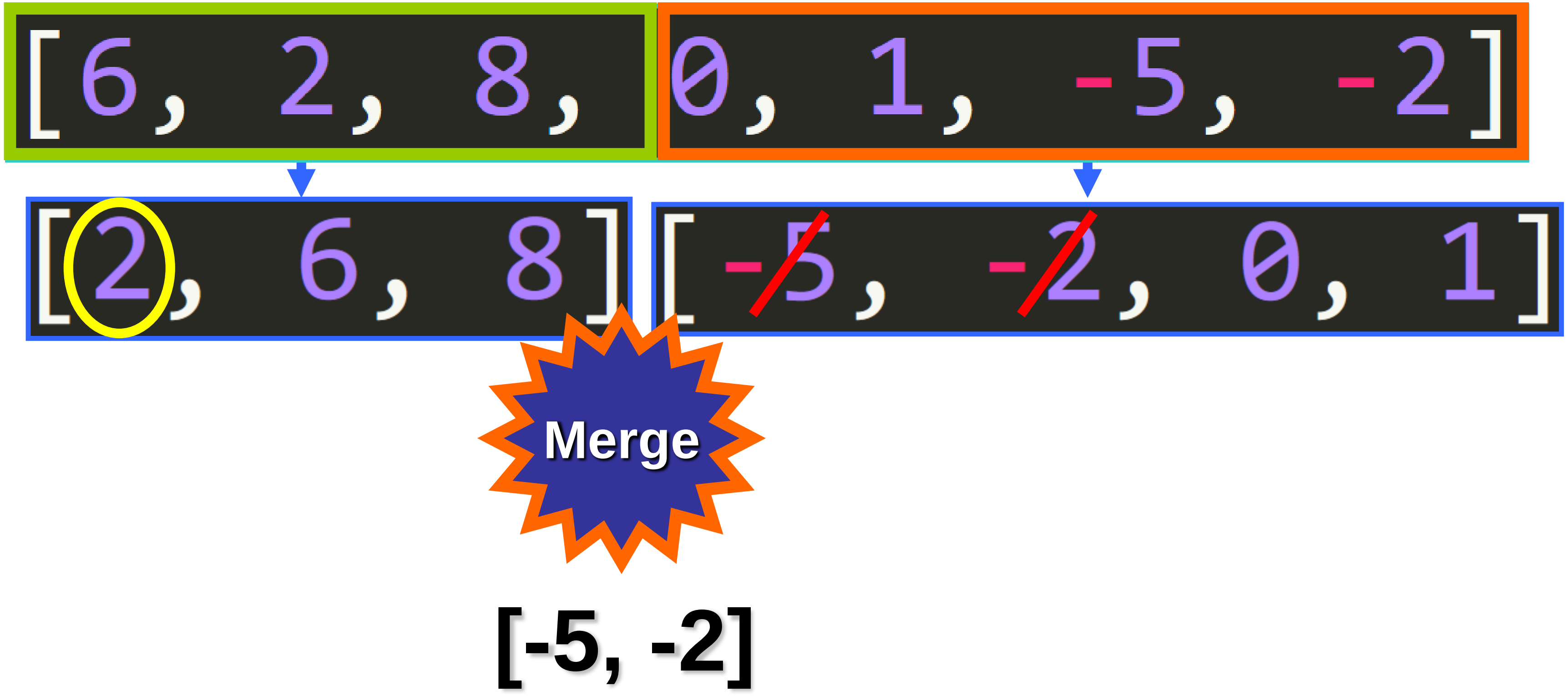
```
[-5, -2, 0, 1]
```

[-5]

Merge Sort



Merge Sort



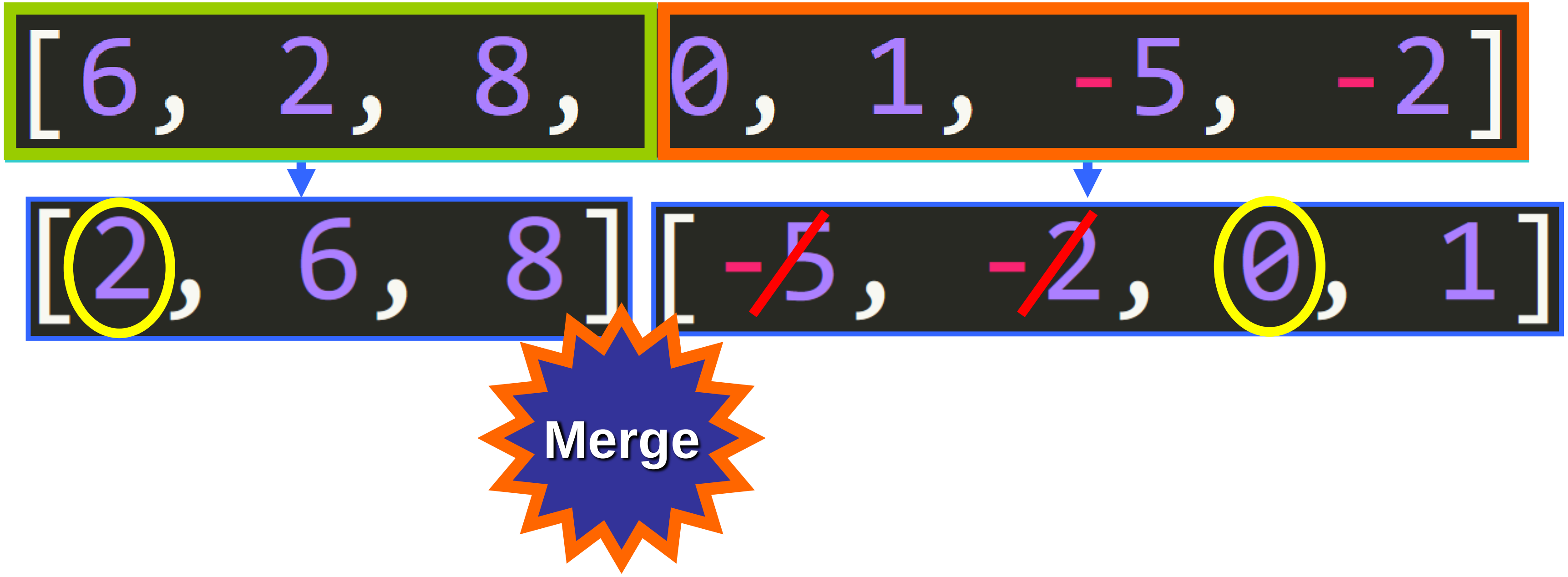
```
[6, 2, 8, 0, 1, -5, -2]
```

```
[2, 6, 8]
```

```
[-5, -2, 0, 1]
```

[-5, -2]

Merge Sort

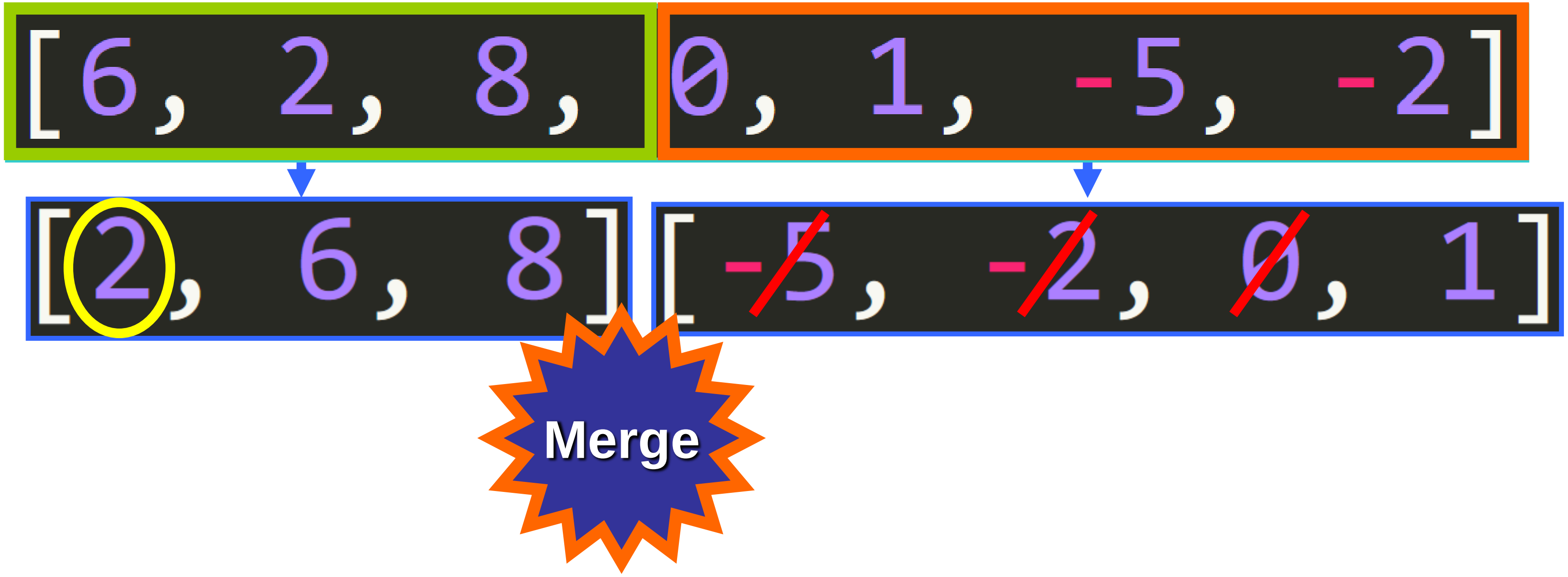


[6, 2, 8, 0, 1, -5, -2]

[2, 6, 8] [~~-5~~, ~~-2~~, 0, 1]

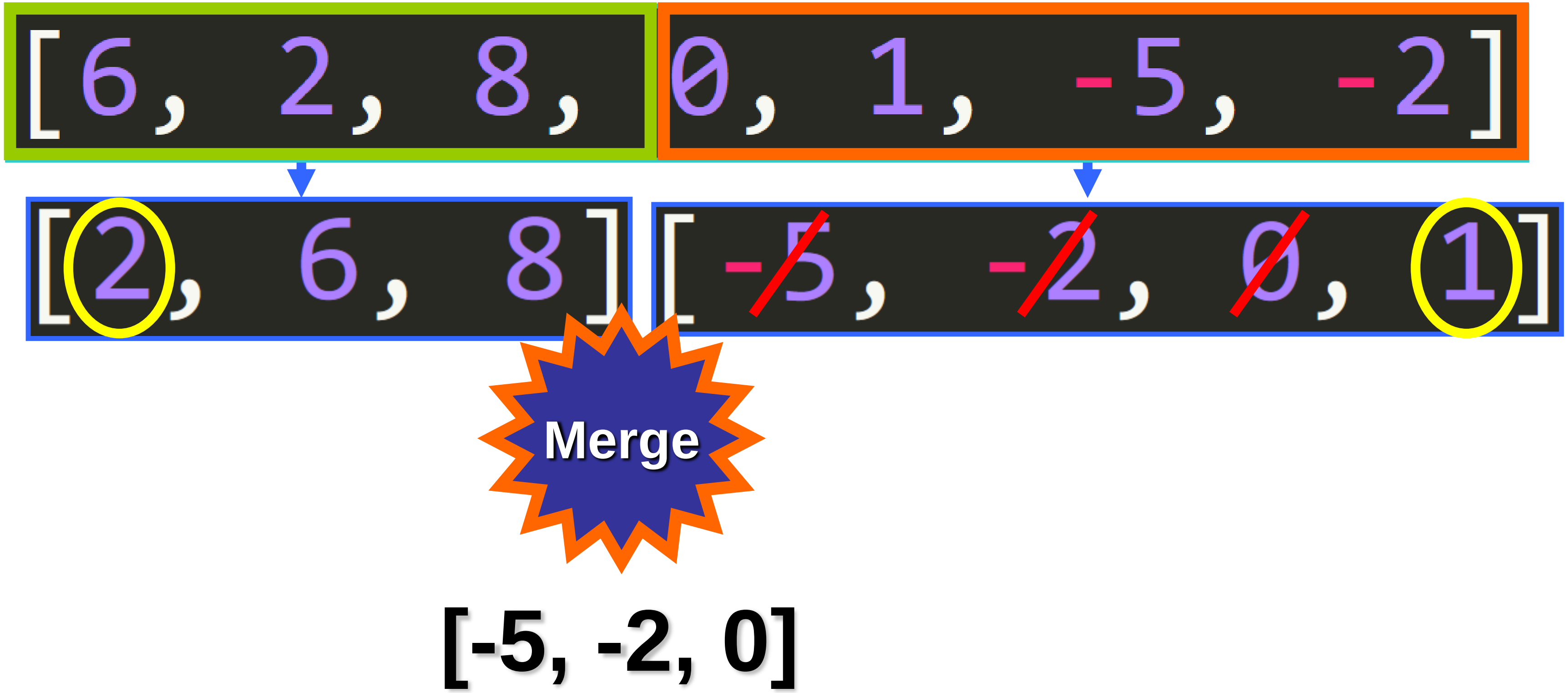
[-5, -2]

Merge Sort

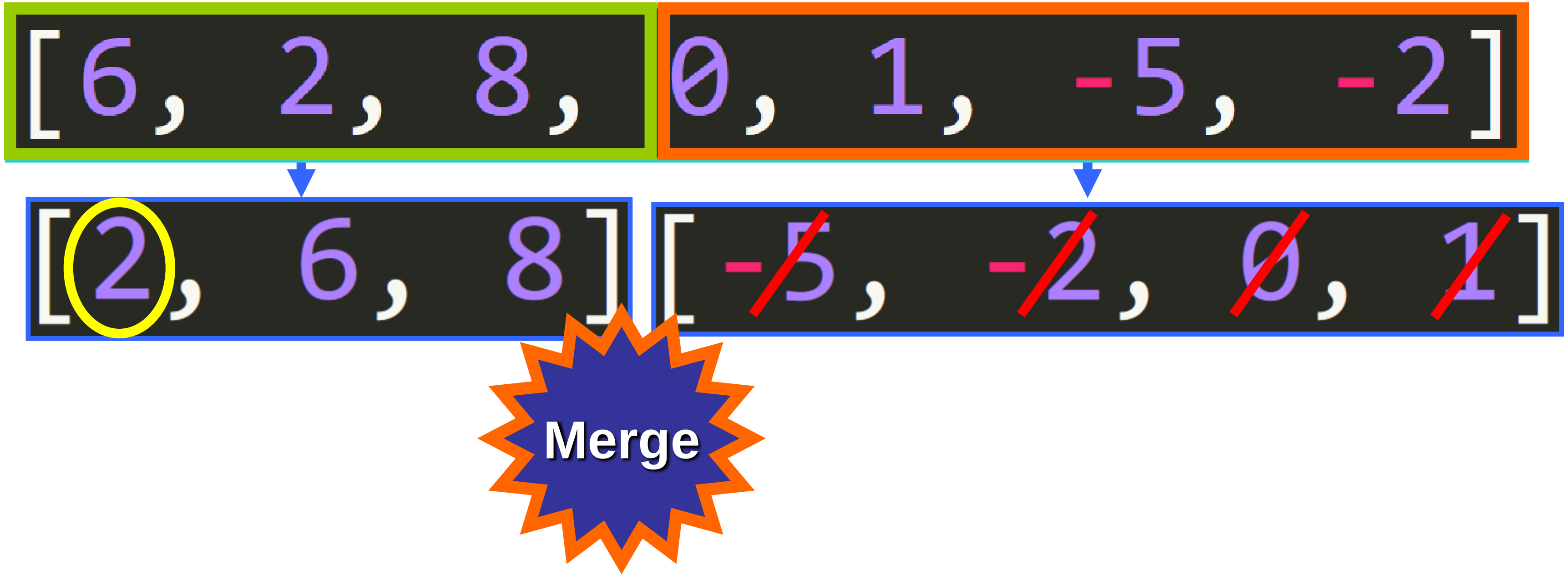


[-5, -2, 0]

Merge Sort

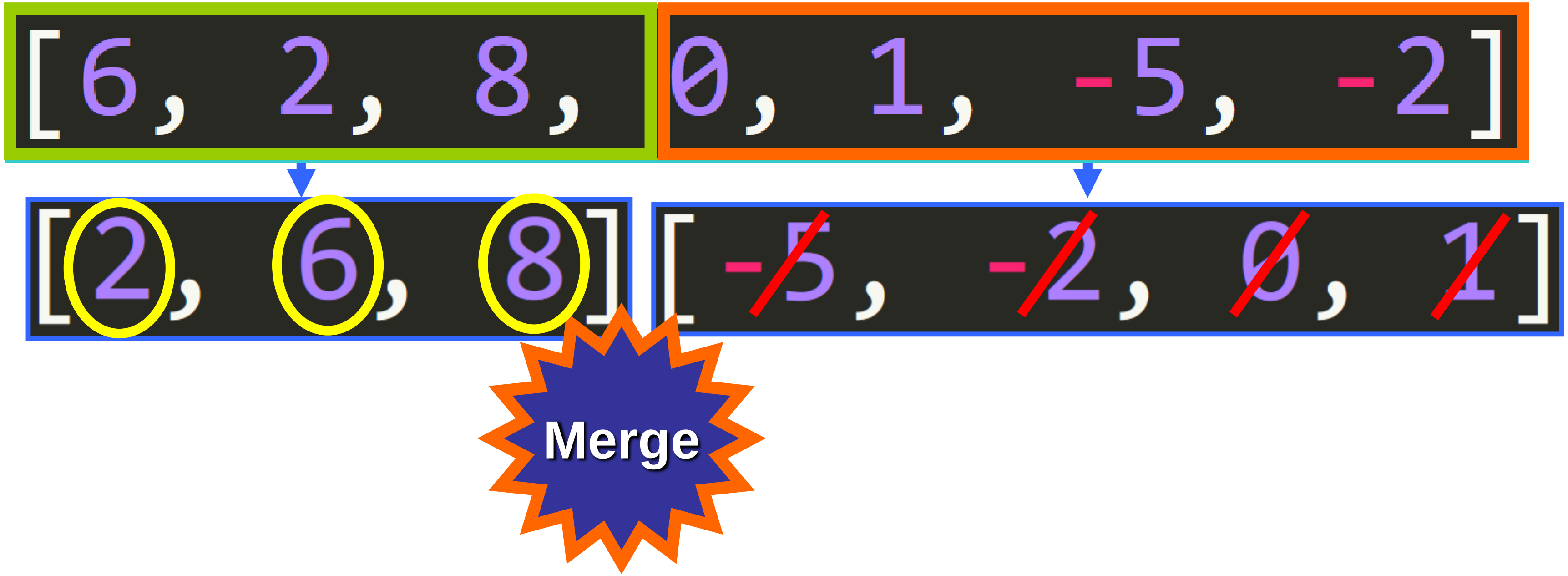


Merge Sort



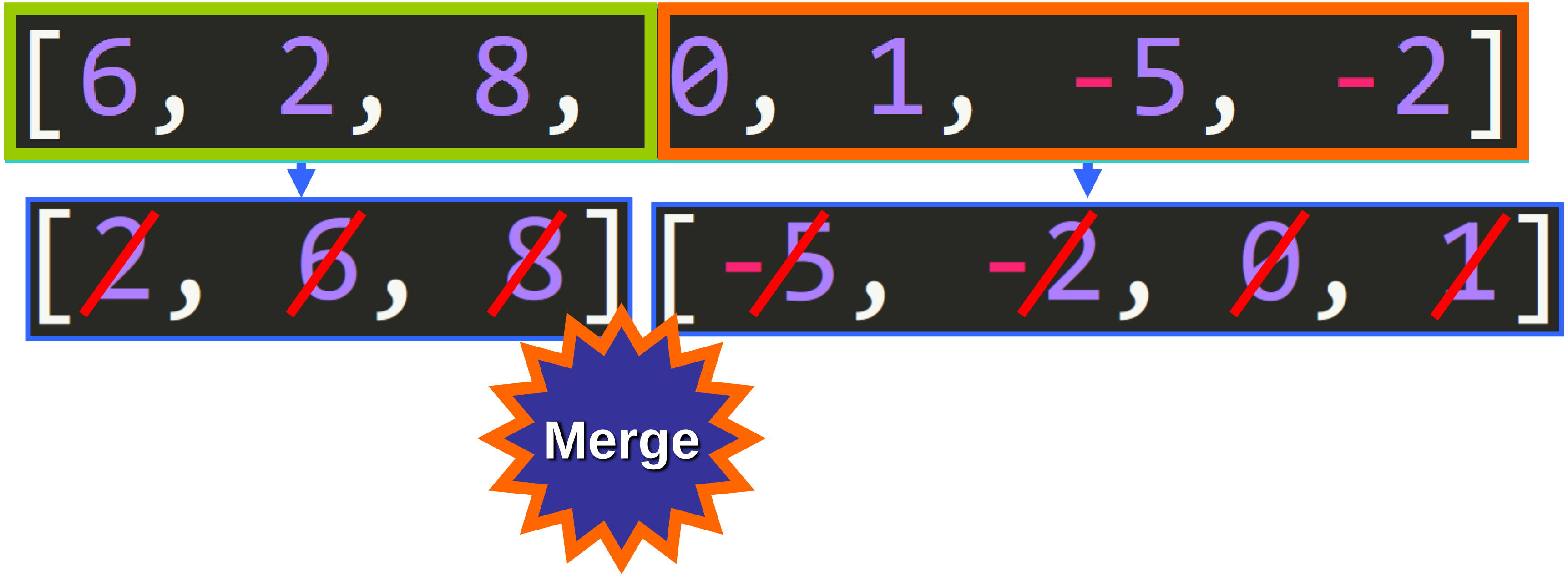
[-5, -2, 0, 1]

Merge Sort



[-5, -2, 0, 1]

Merge Sort



[-5, -2, 0, 1, 2, 6, 8]

Merge Sort

```
[-5, -2, 0, 1, 2, 6, 8]
```

Sorted!

To the Code!

Algorithm

Merge Sort

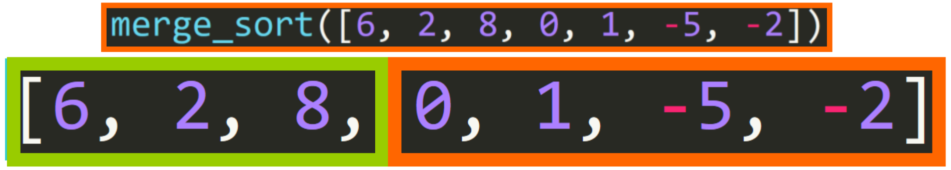
Example | Code

```python
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([6, 2, 8, 0, 1, -5, -2])
```

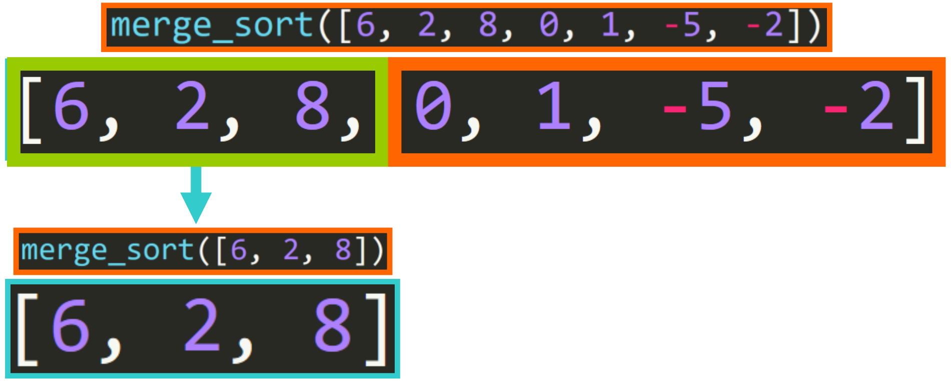
```
[6, 2, 8, 0, 1, -5, -2]
```

```python
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)
```

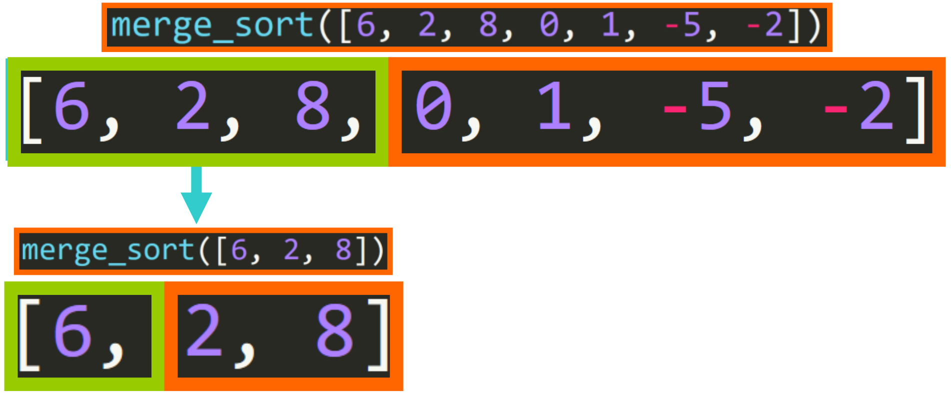


```
merge_sort([6, 2, 8, 0, 1, -5, -2])
[6, 2, 8, 0, 1, -5, -2]
```

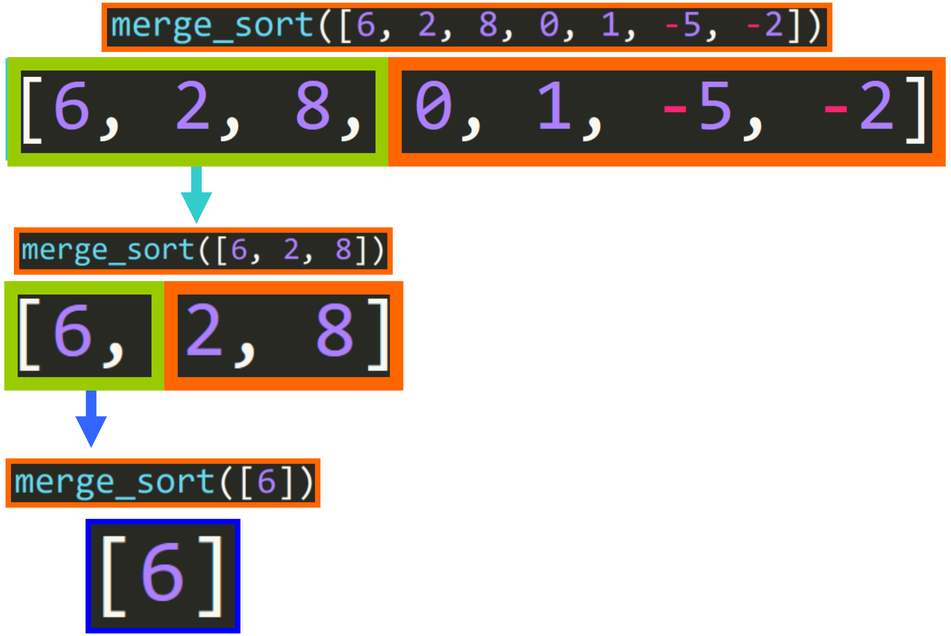
```python
def merge_sort(lst):

    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)
```



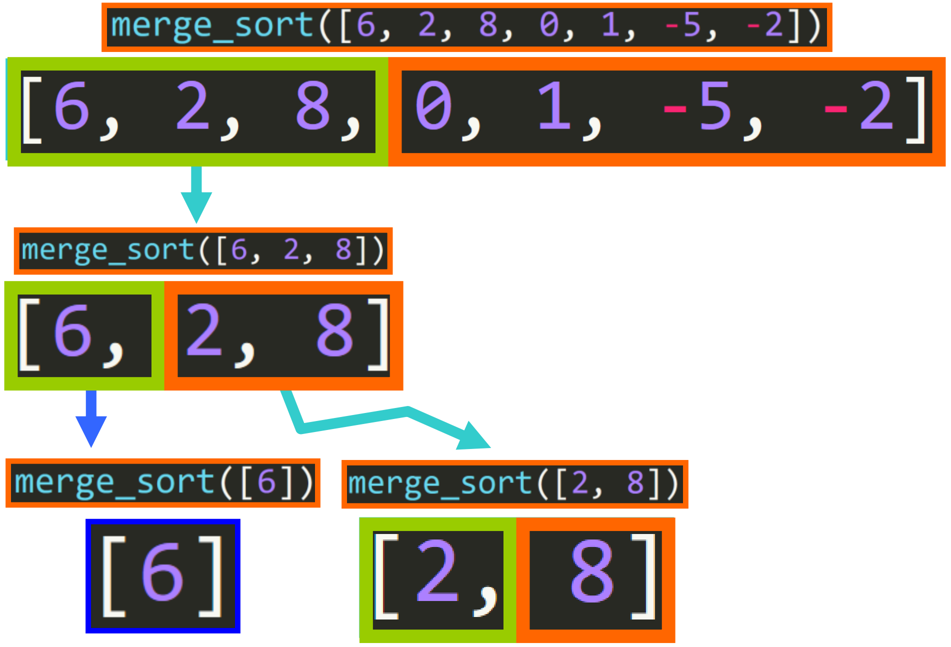
```python
def merge_sort(lst):

    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)
```



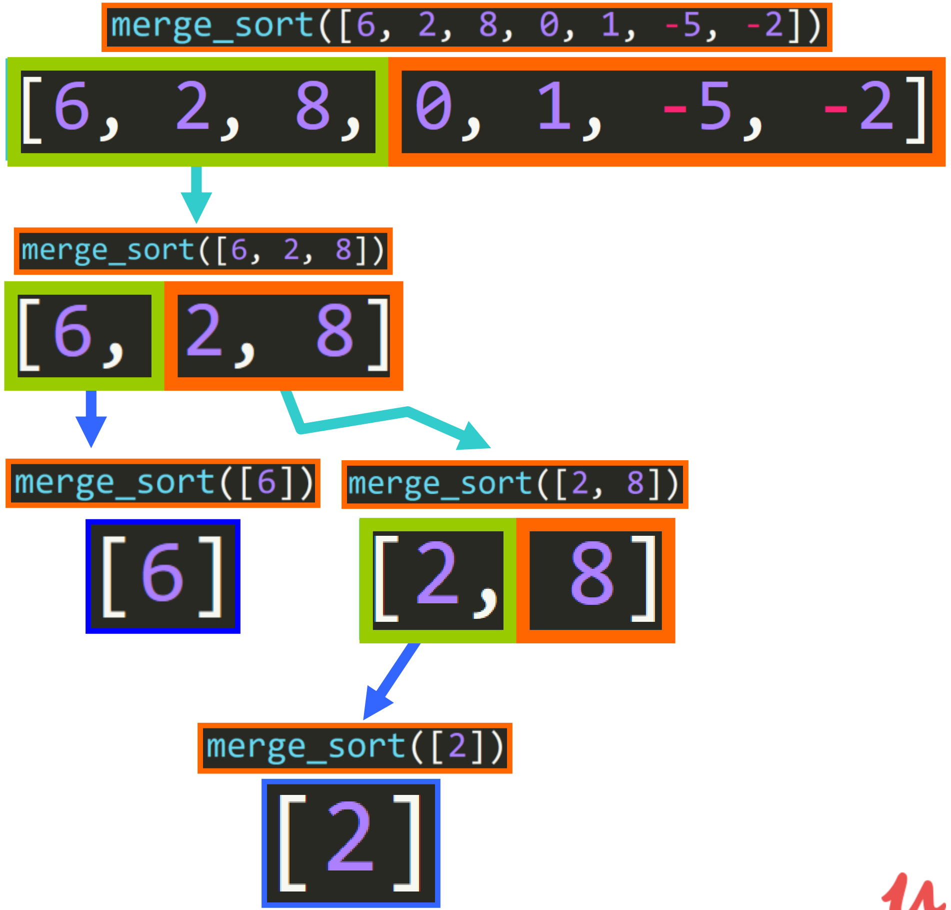
```python
def merge_sort(lst):

    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)
```



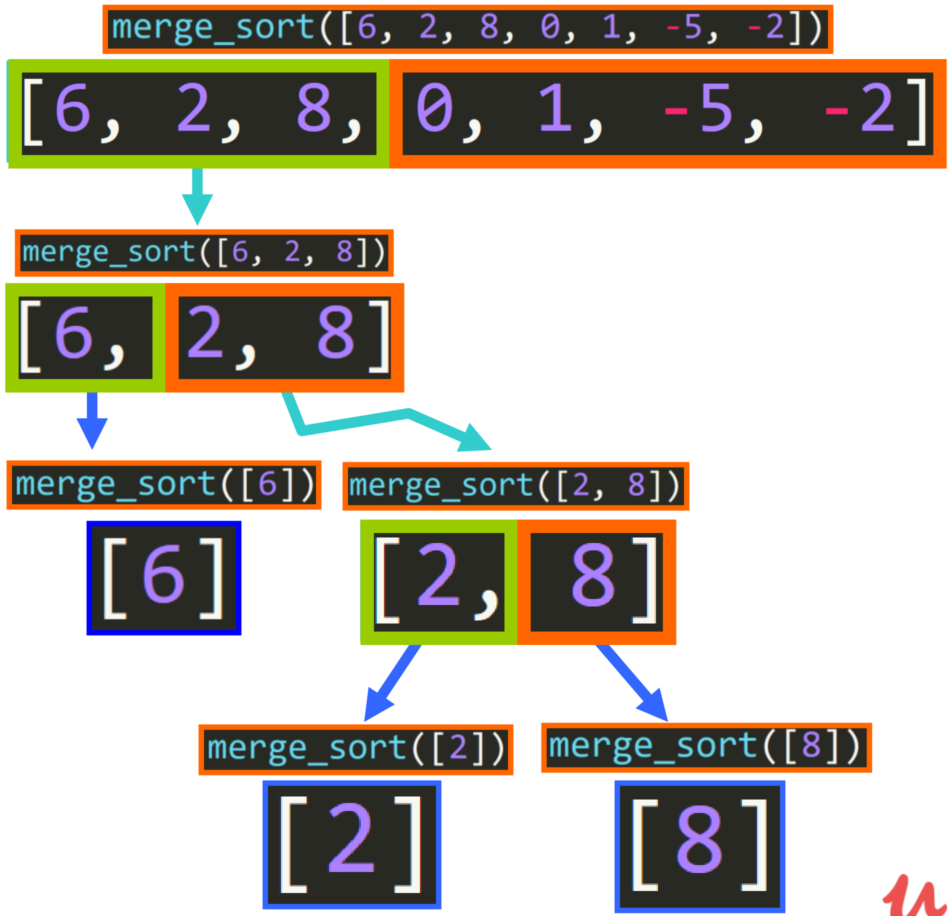
```python
def merge_sort(lst):

    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)
```



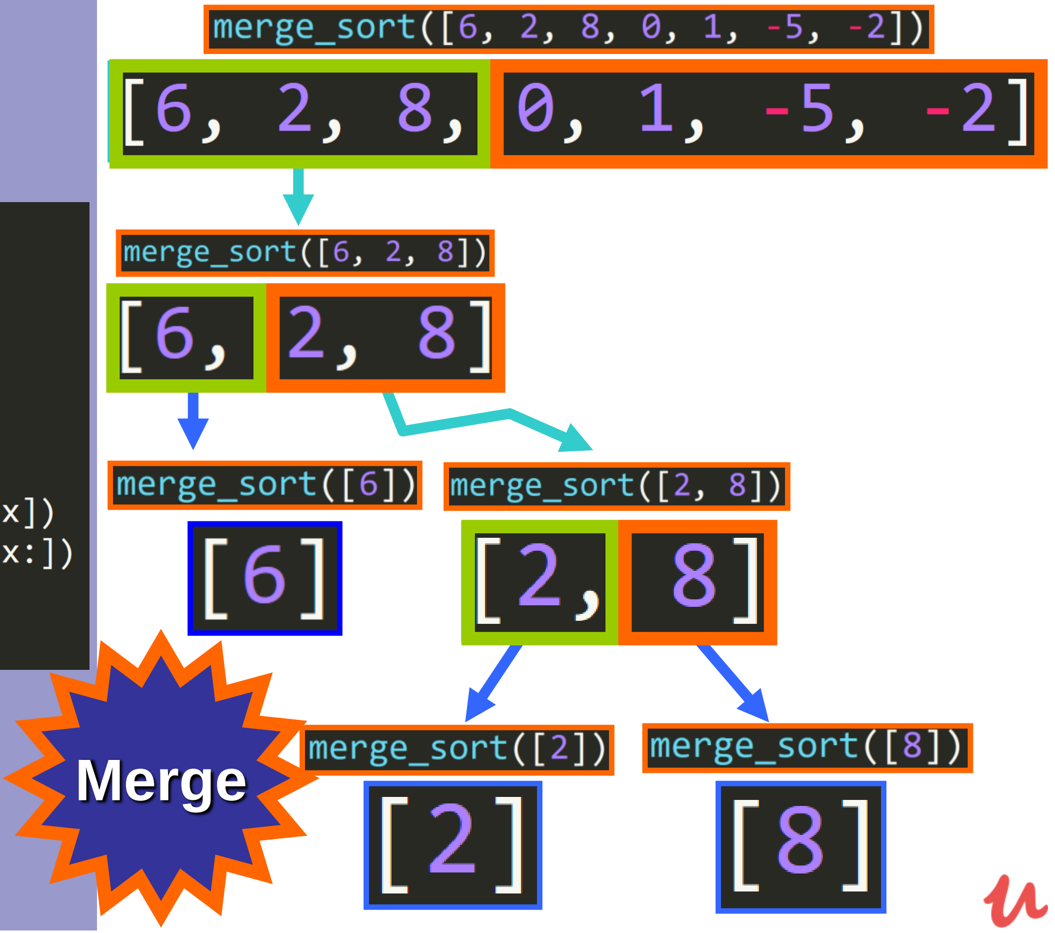
```python
def merge_sort(lst):

    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)
```

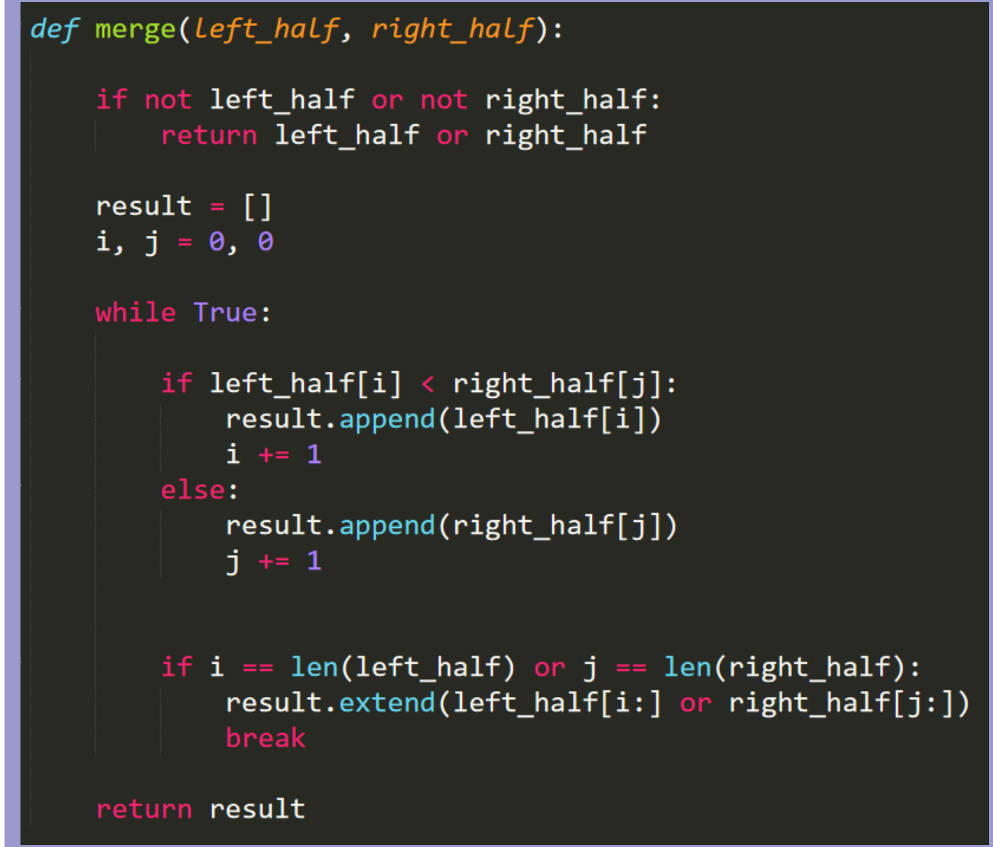


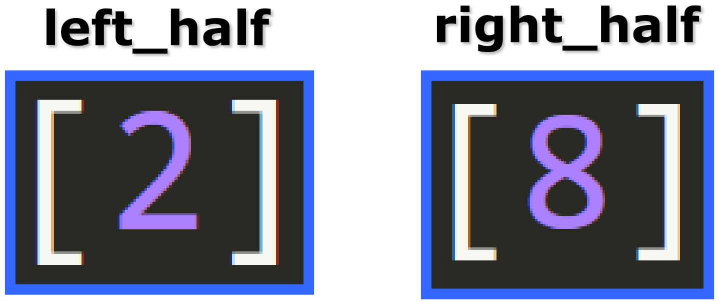
```python
def merge_sort(lst):

    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)
```

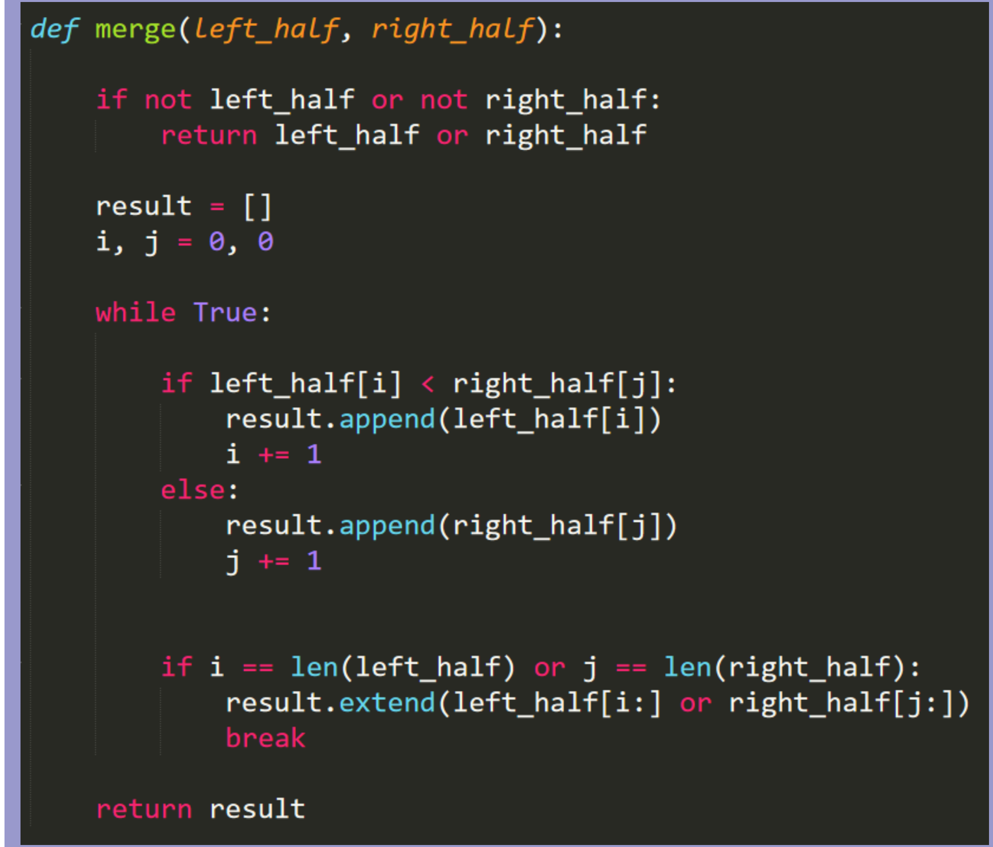


```python
def merge_sort(lst):

    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)
```




```python
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]:
            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
```

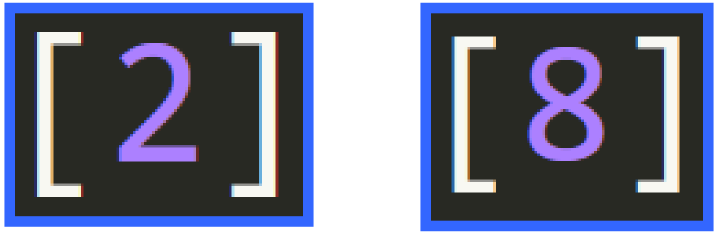




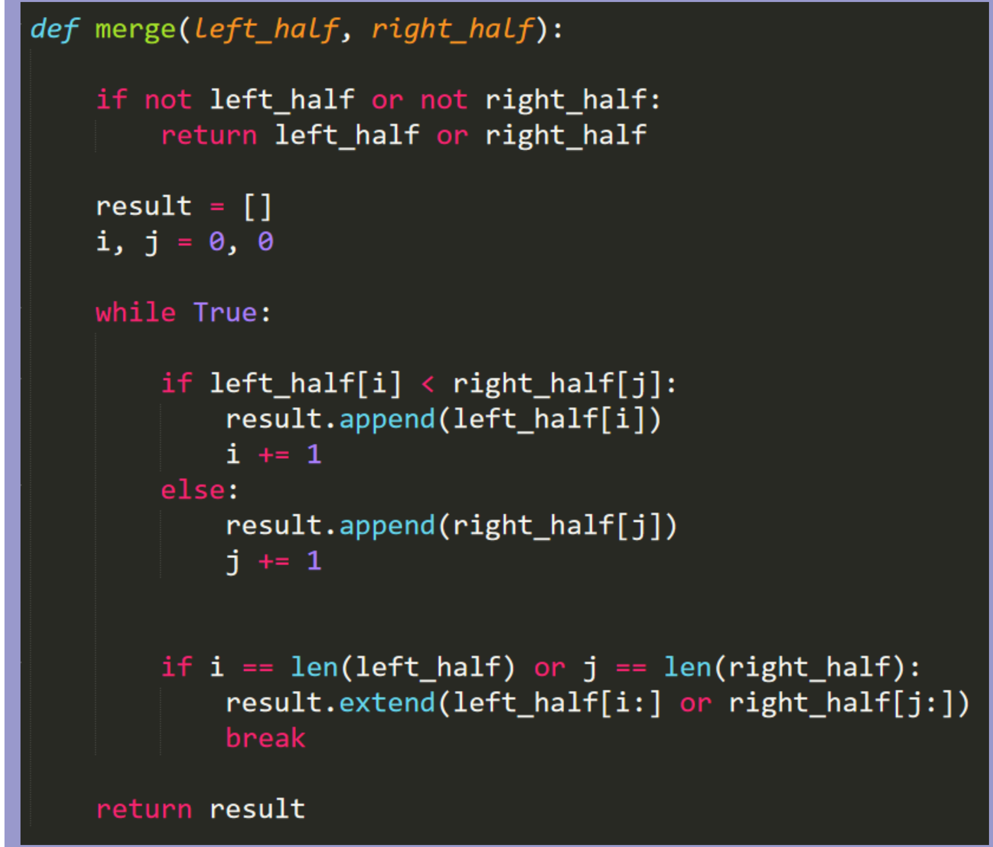
```
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 0

left_half   **right_half**

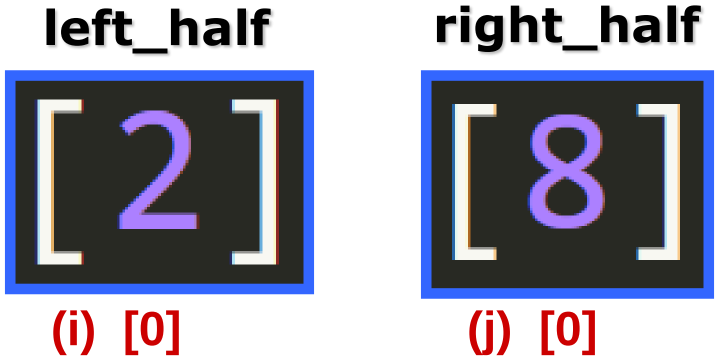


result = [ ]

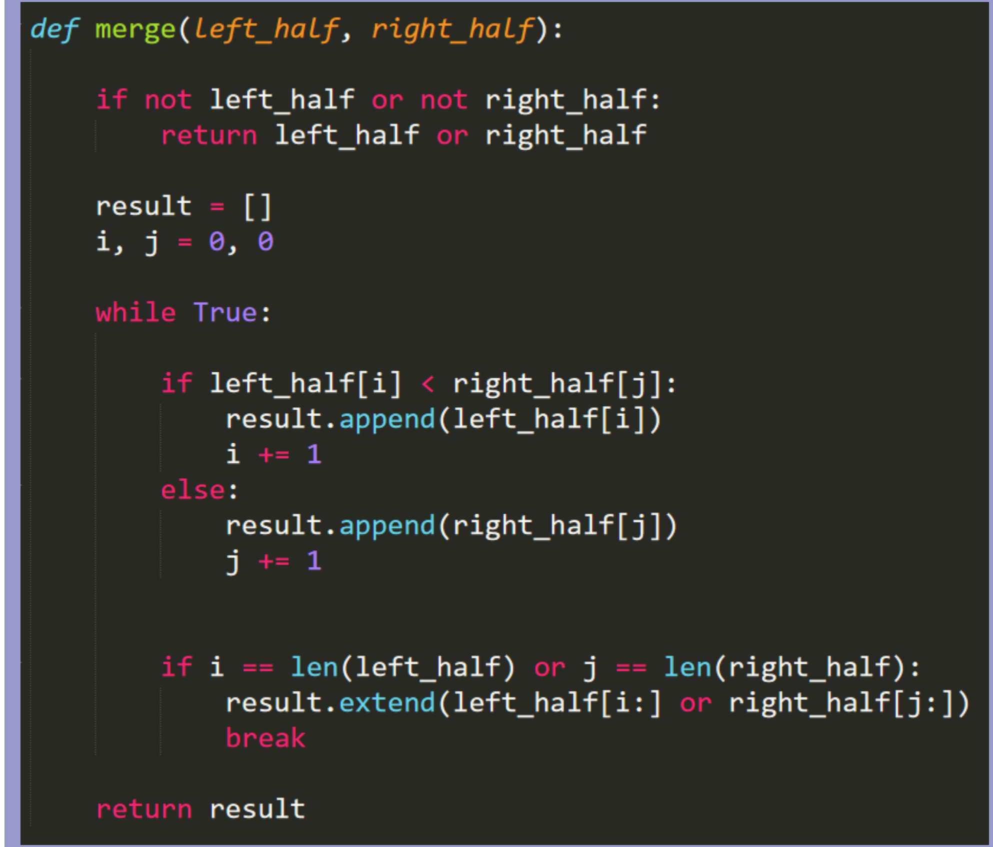


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 0

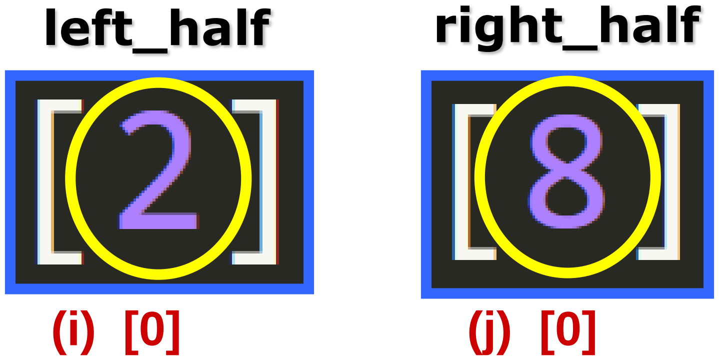


result = [ ]



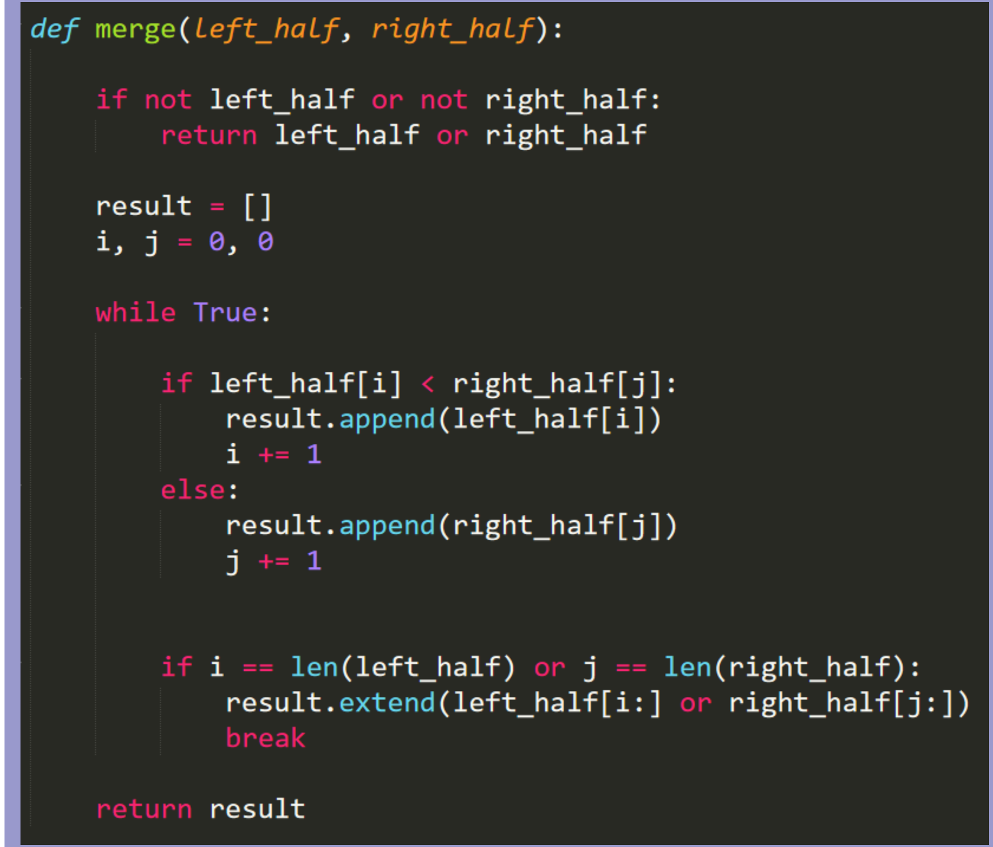
```
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 0



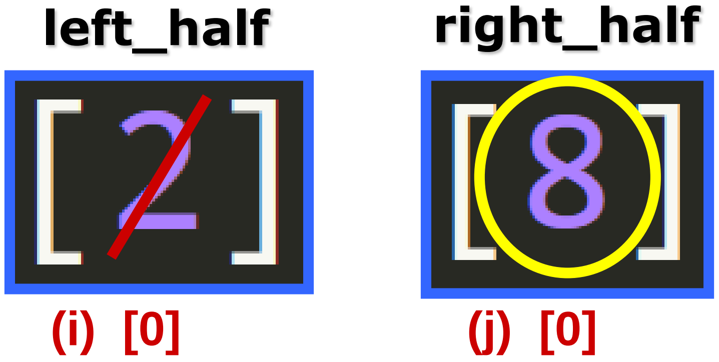
2 < 8?

result = [ ]

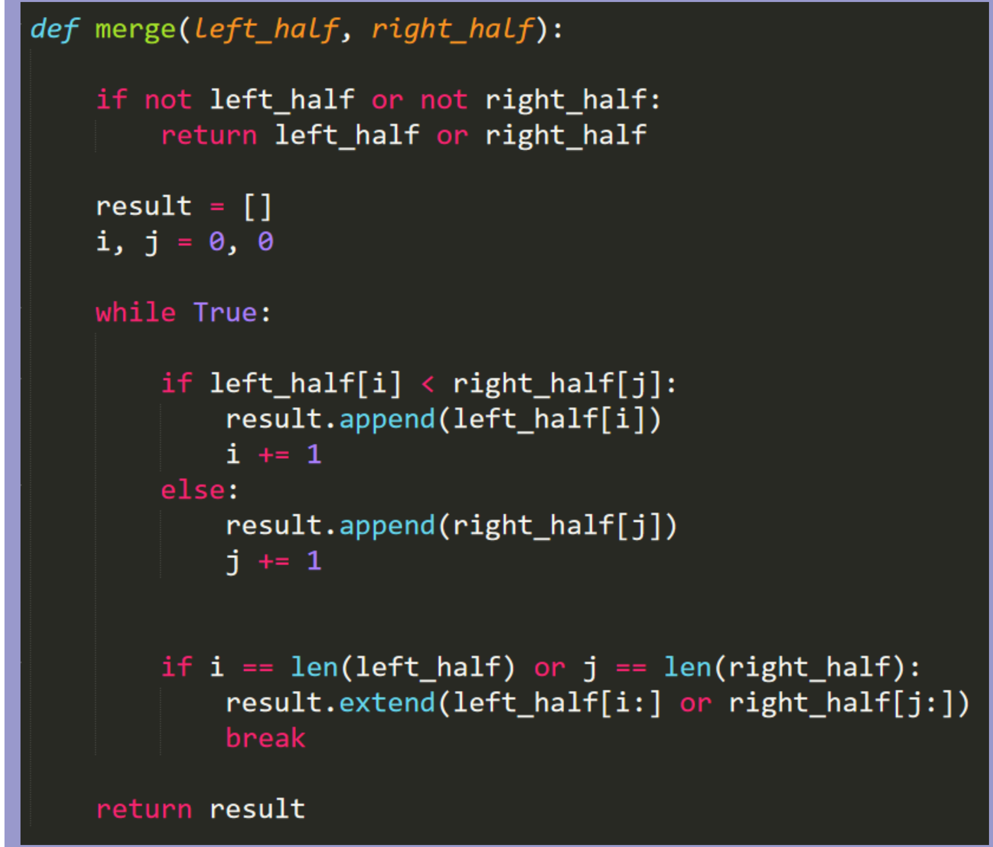


```
def merge(left_half, right_half):

    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]:
            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
```

i = 1   j = 0



result = [2]



```
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]:
            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
```

i = 1   j = 0

left_half

$[\cancel{2}]$

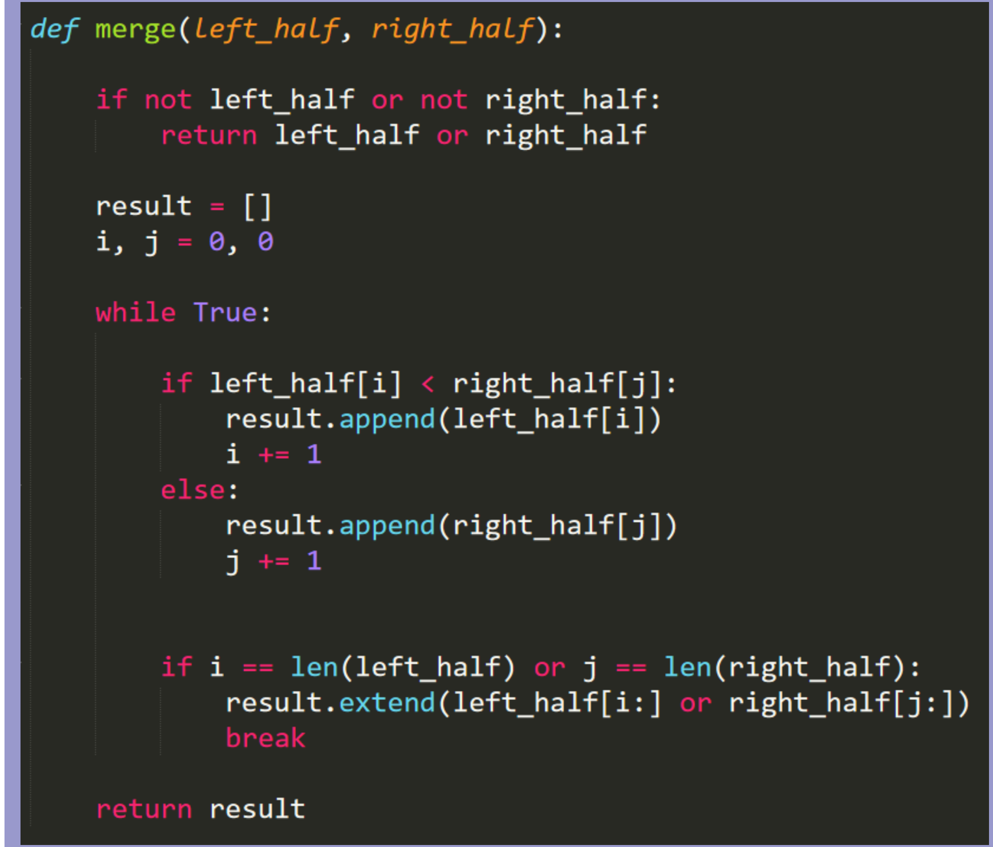
(i) [0]

right_half

$[\cancel{8}]$

(j) [0]

result = [2, 8]



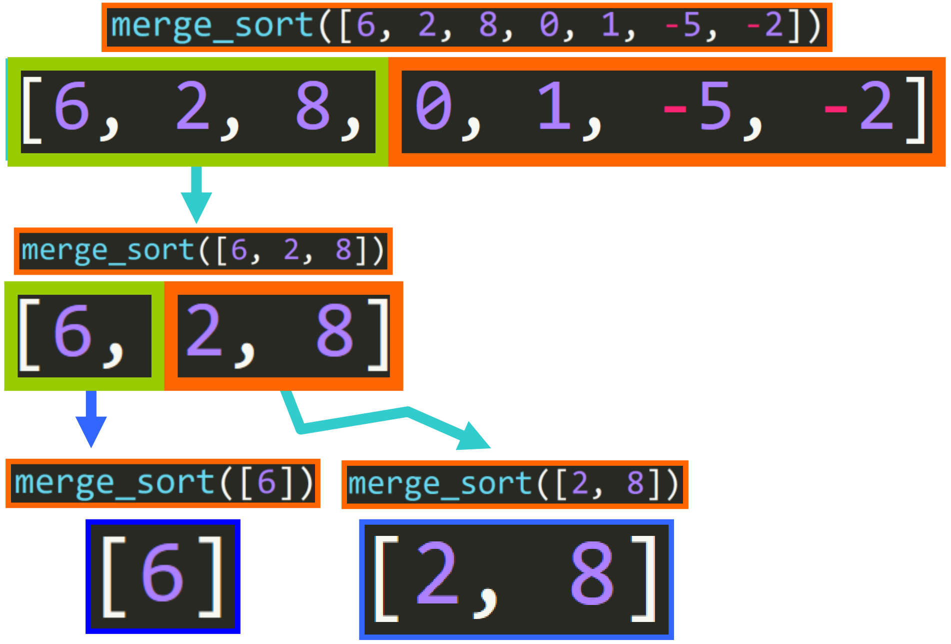
```
def merge(left_half, right_half):

    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]:
            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
```

i = 1 j = 0

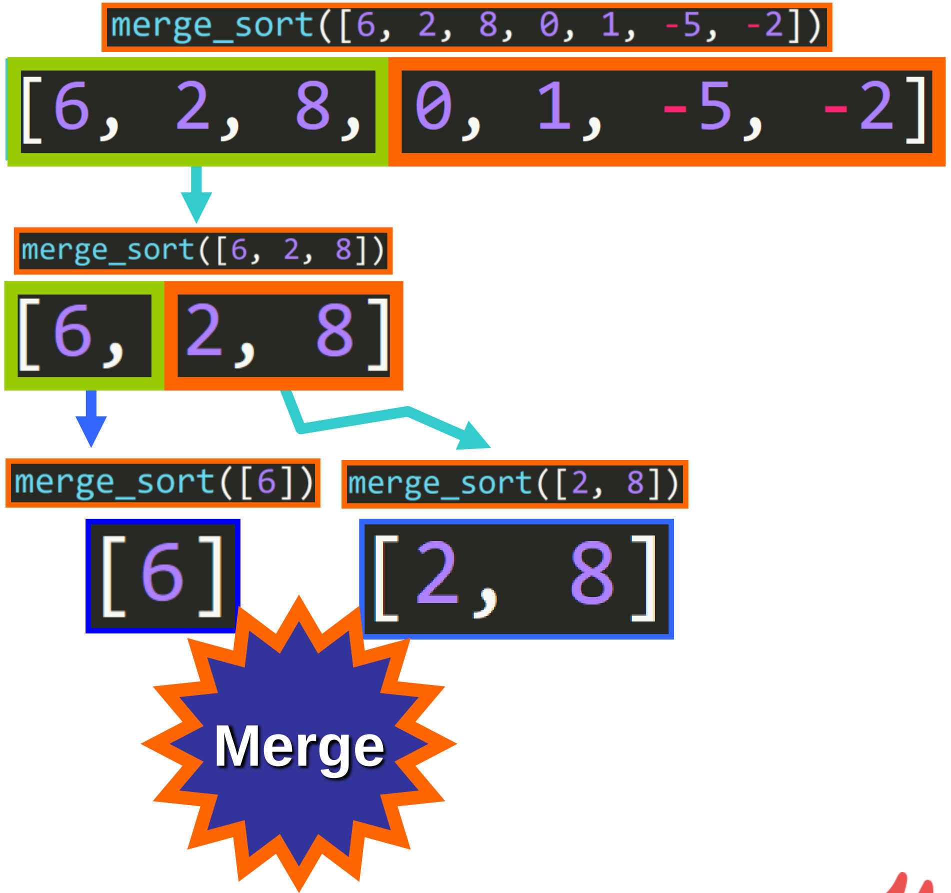
[2, 8]

result = [2, 8]

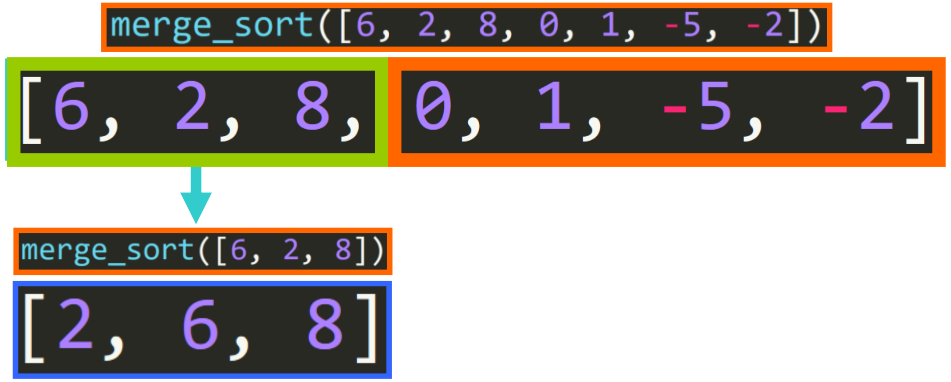
```python
def merge_sort(lst):

    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)
```



```python
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)
```

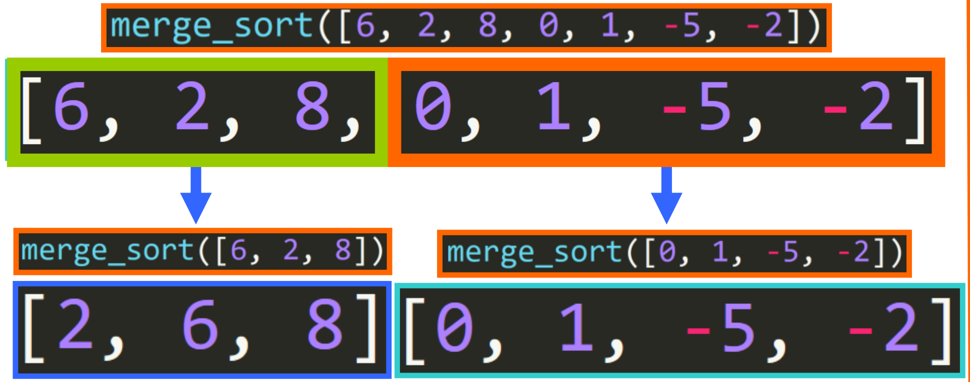


```python
def merge_sort(lst):

    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)
```

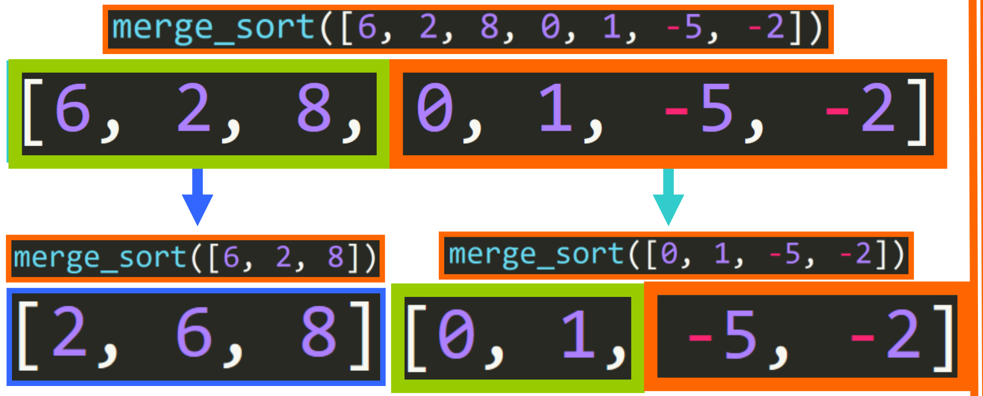


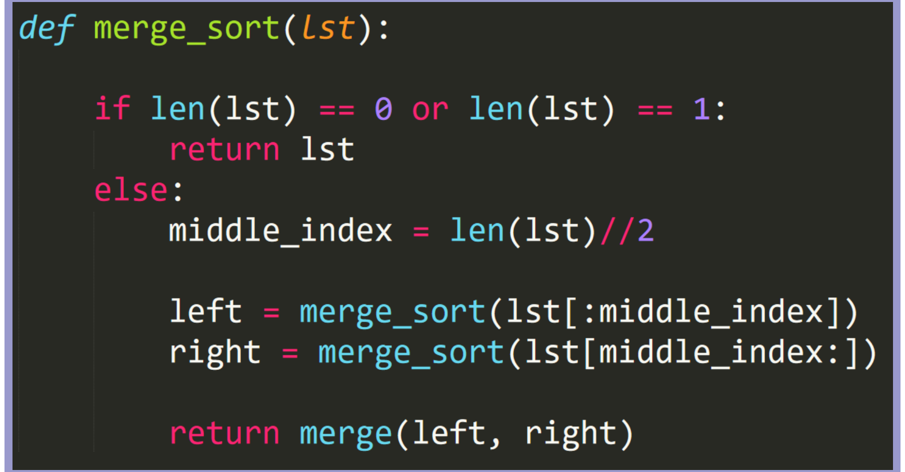
```
merge_sort([6, 2, 8, 0, 1, -5, -2])
[6, 2, 8, 0, 1, -5, -2]
merge_sort([6, 2, 8])
[2, 6, 8]
```

```python
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)
```

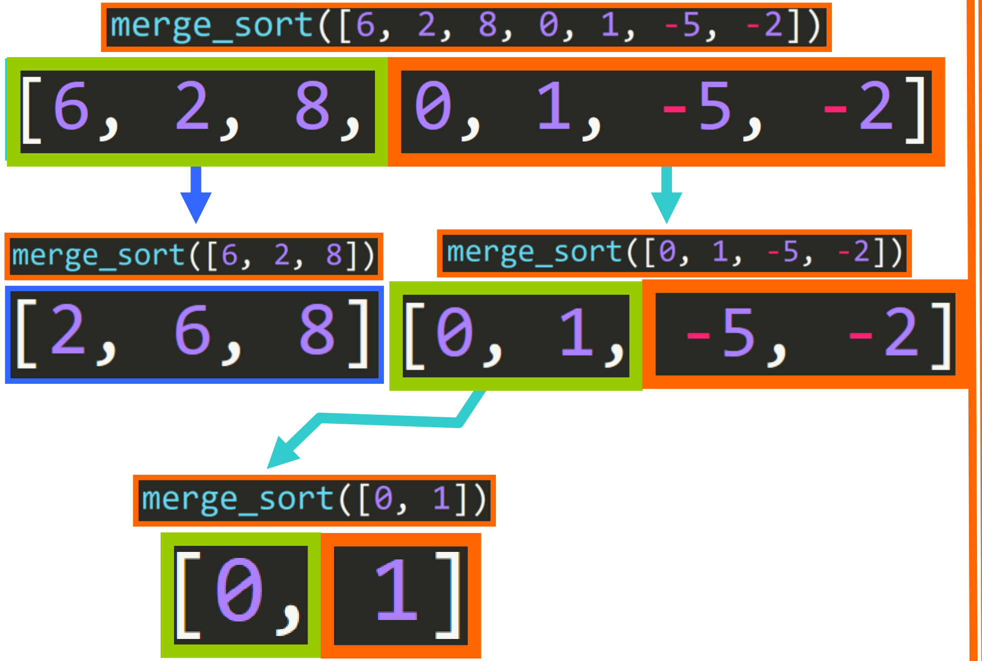


```python
def merge_sort(lst):

    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)
```

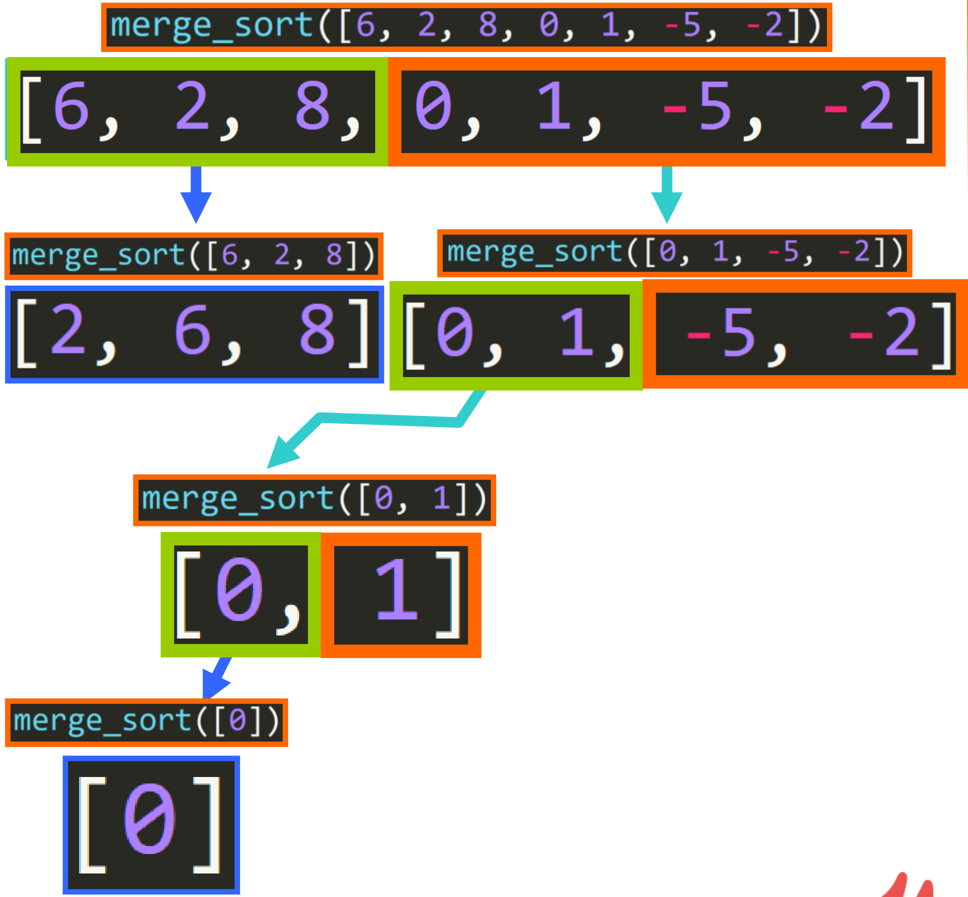




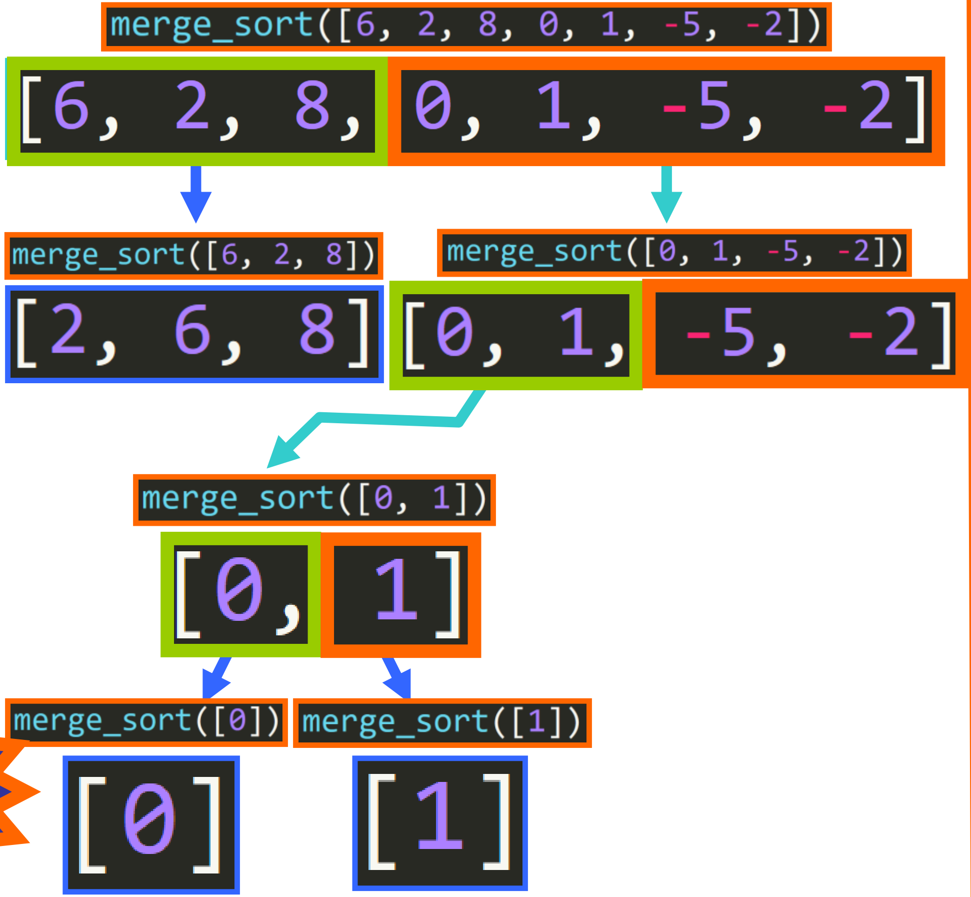
```python
def merge_sort(lst):

    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)
```

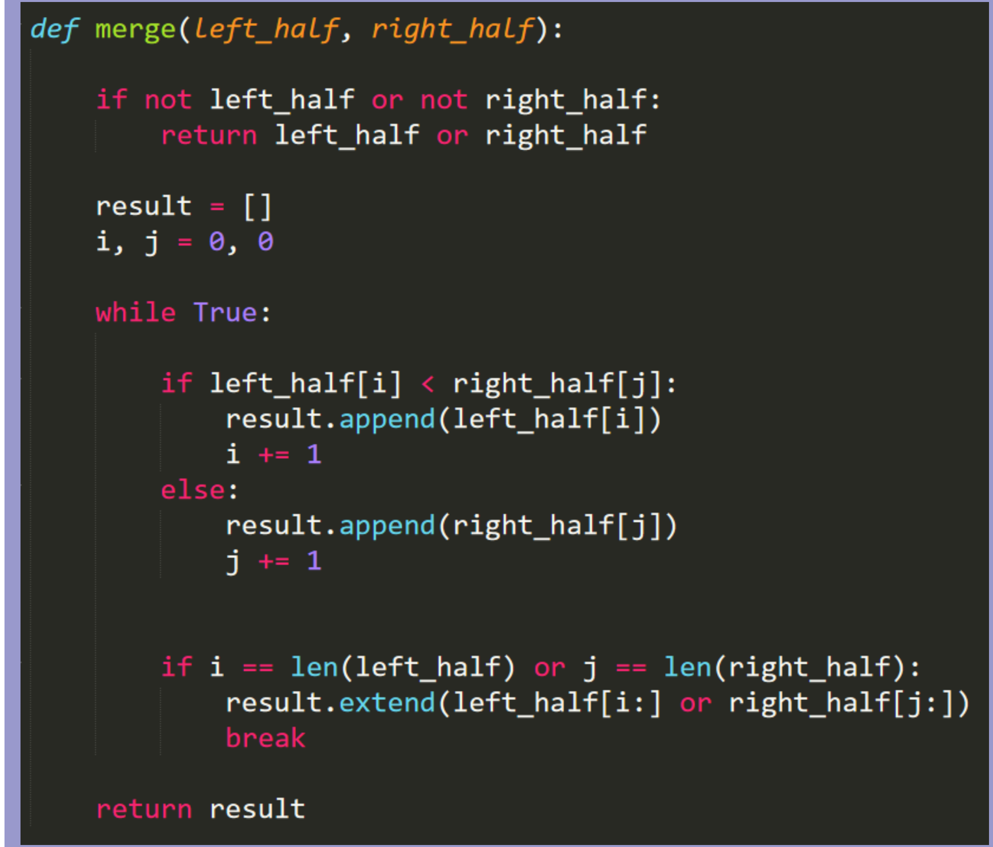


```python
def merge_sort(lst):

    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)
```

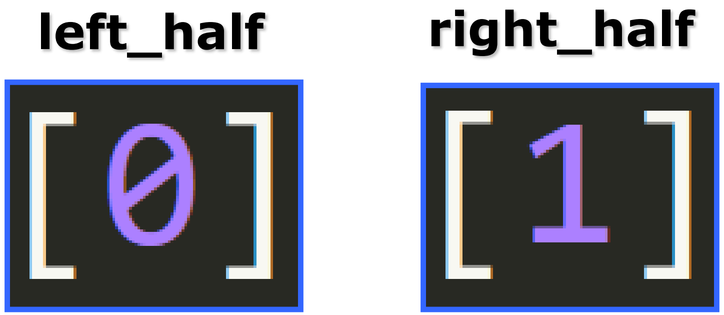


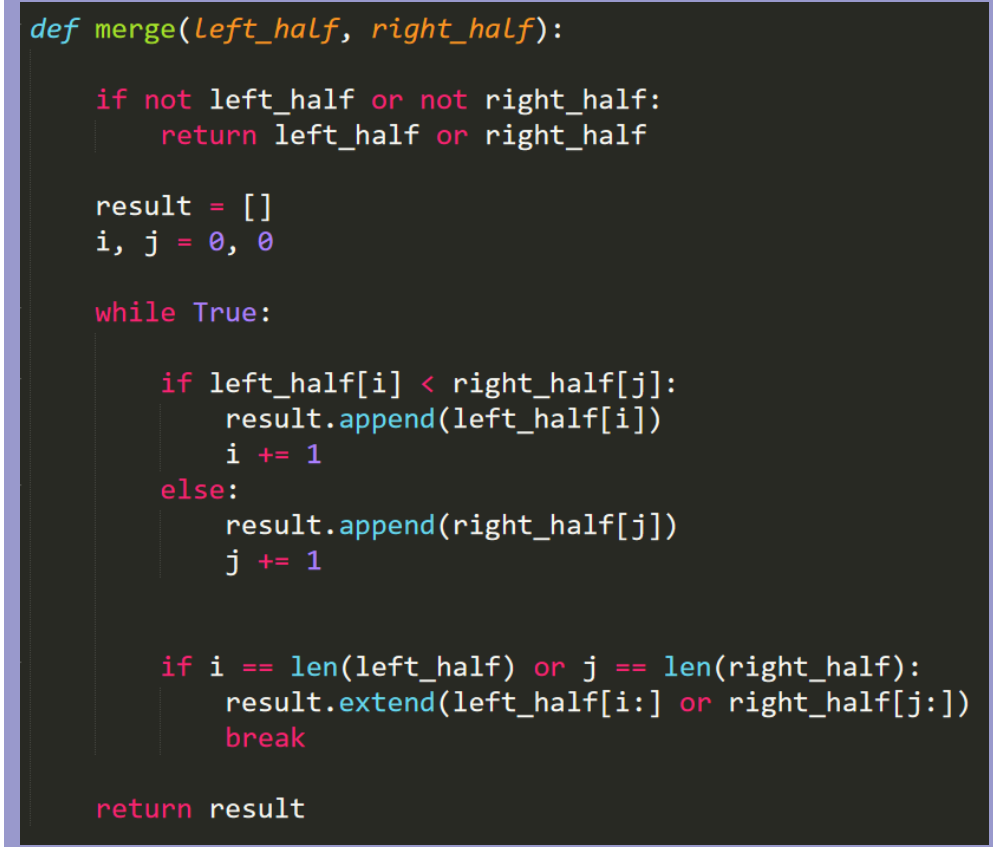
```python
def merge_sort(lst):

    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)
```





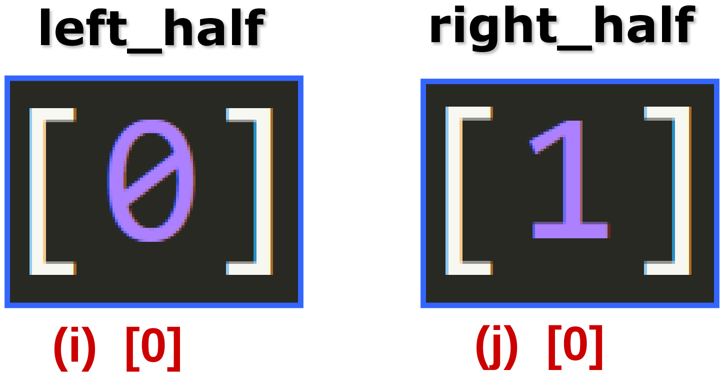
```python
def merge(left_half, right_half):

    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]:
            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
```



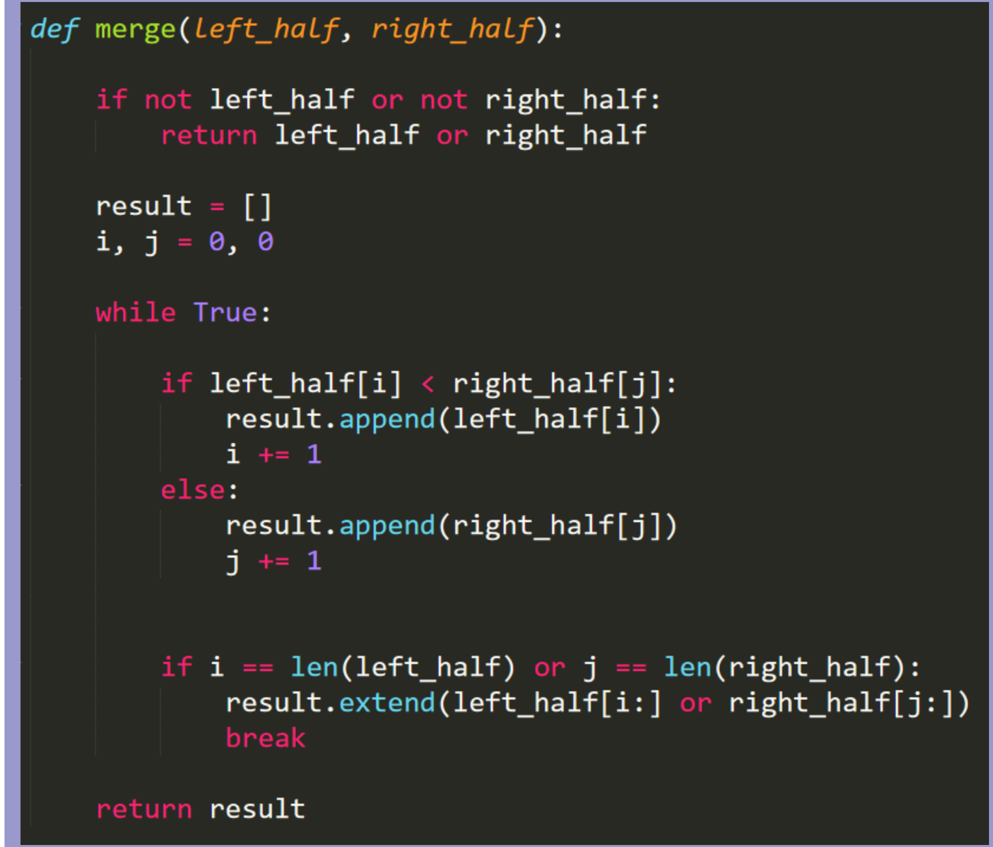


```
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 0

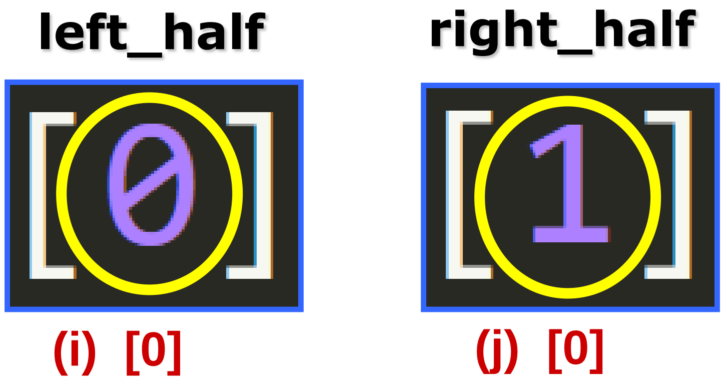


result = [ ]



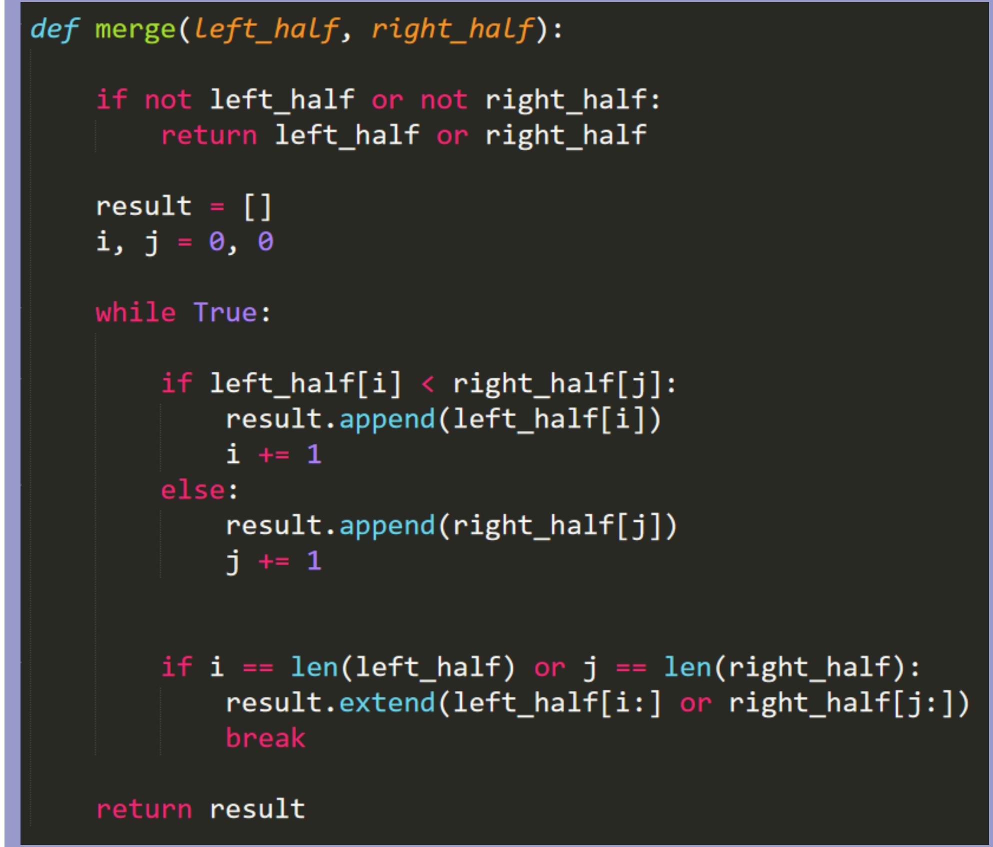
```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 0



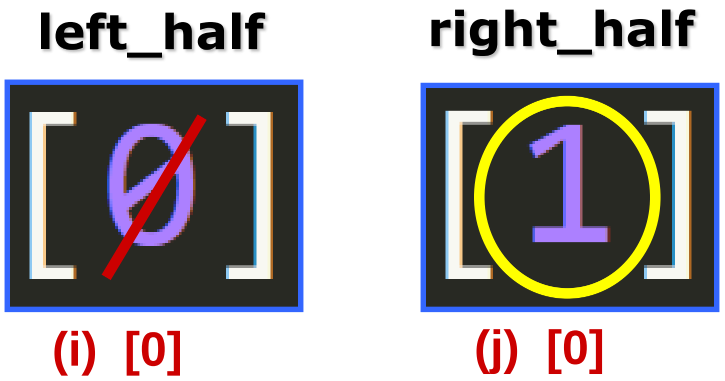
0 < 1?

result = [ ]

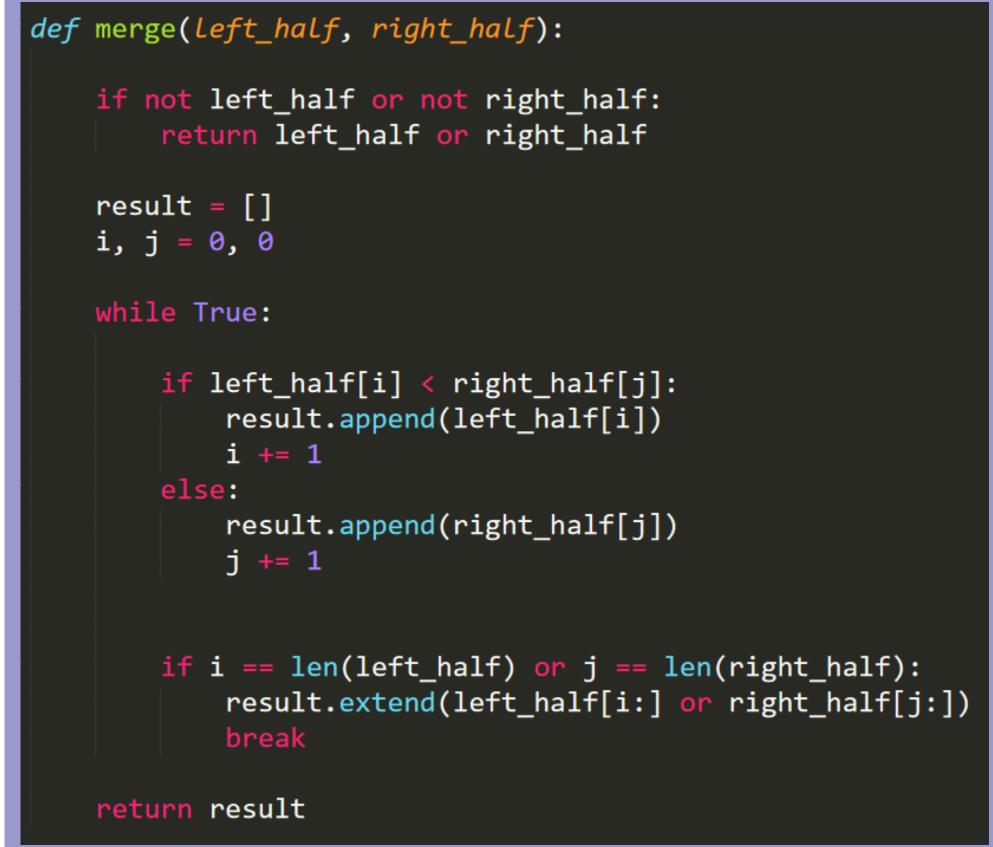


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 1   j = 0



result = [0]

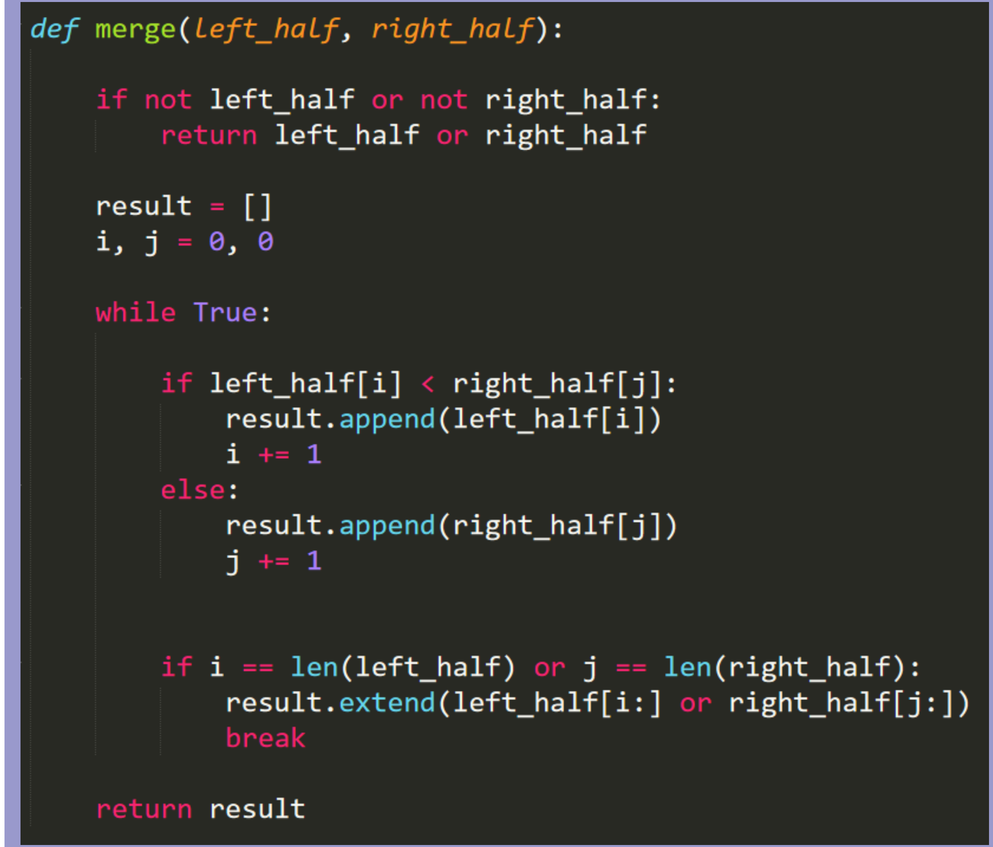


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 1 j = 0

left_half	right_half
[~~0~~]	[~~1~~]
(i) [0]	(j) [0]

result = [0, 1]



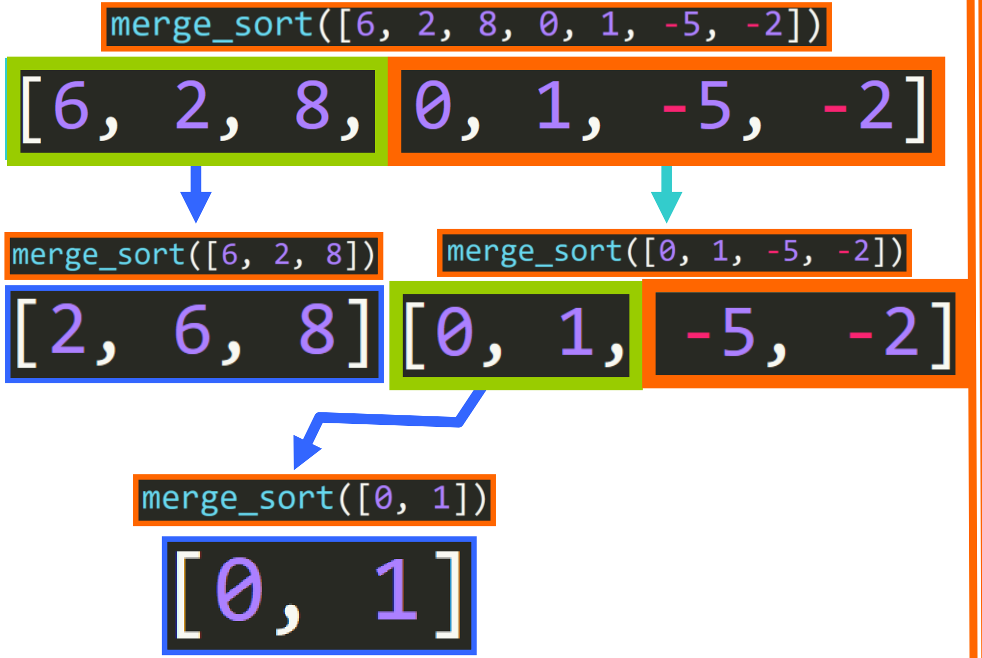
```python
def merge(left_half, right_half):

    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]:
            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
```

i = 1   j = 0

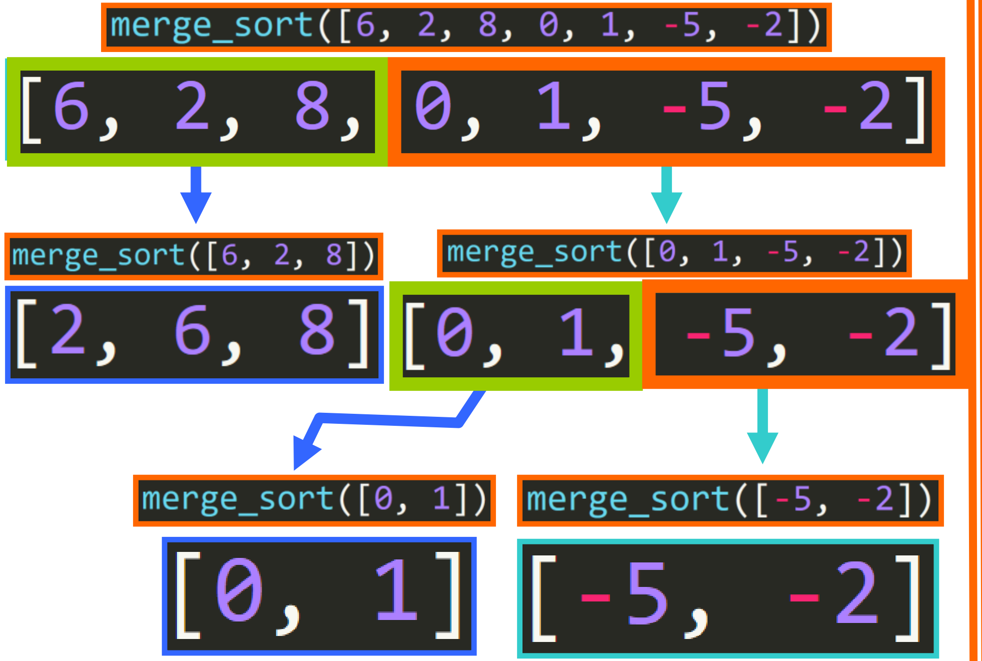
```
[0, 1]
```

result = [0, 1]

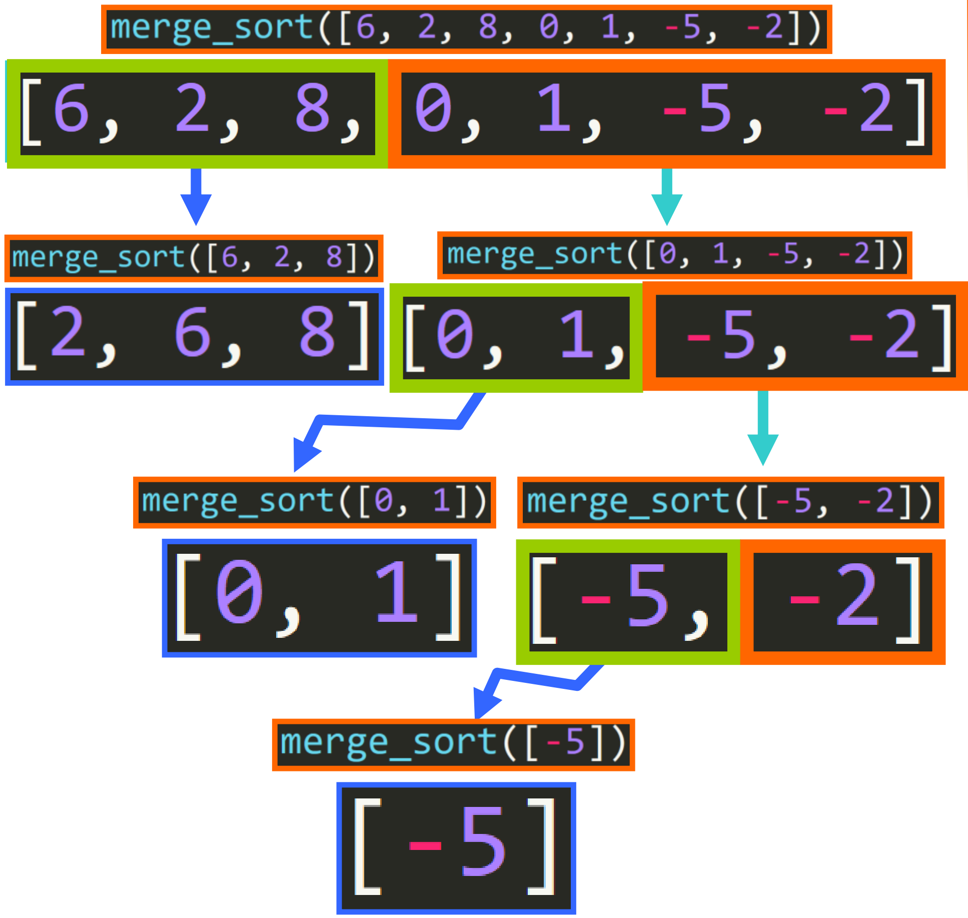
```python
def merge_sort(lst):

    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)
```

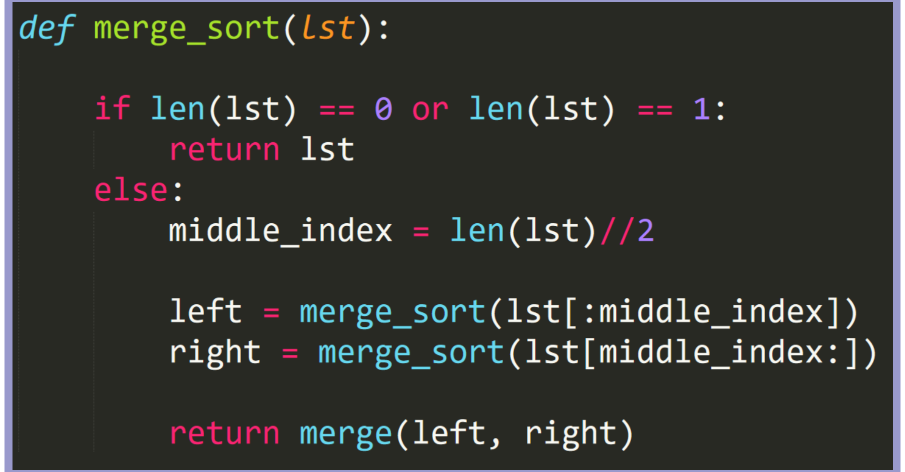


```python
def merge_sort(lst):

    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)
```

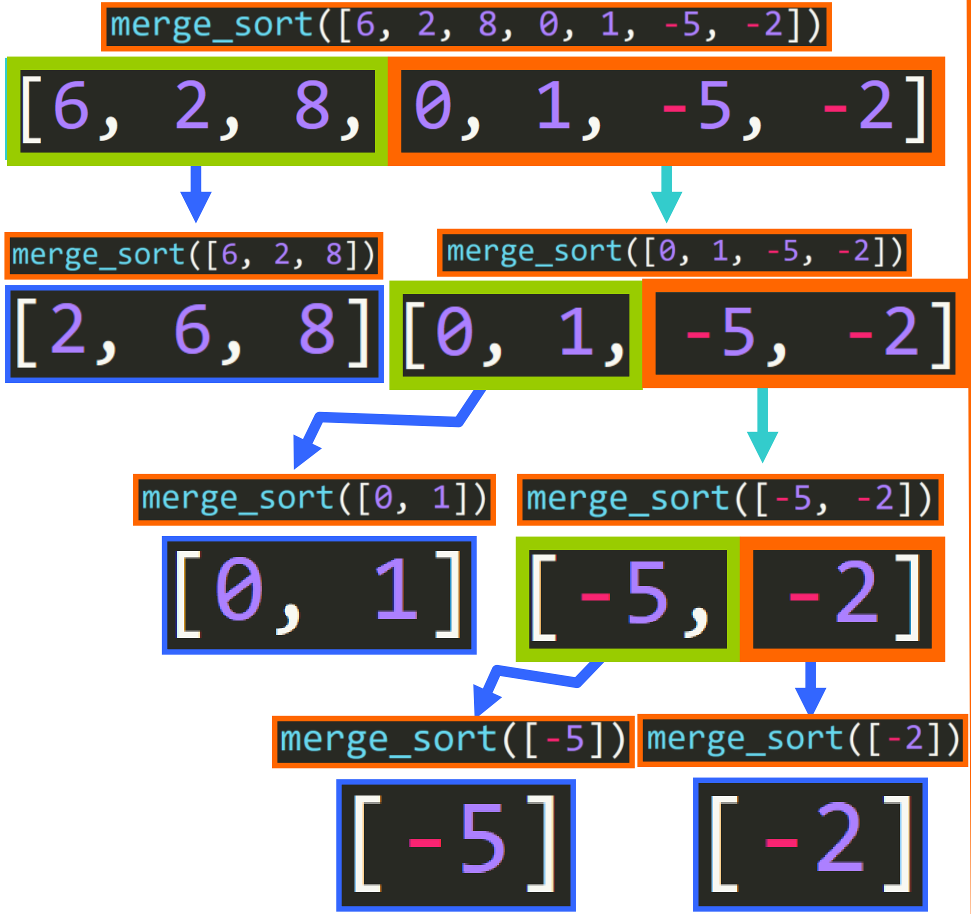


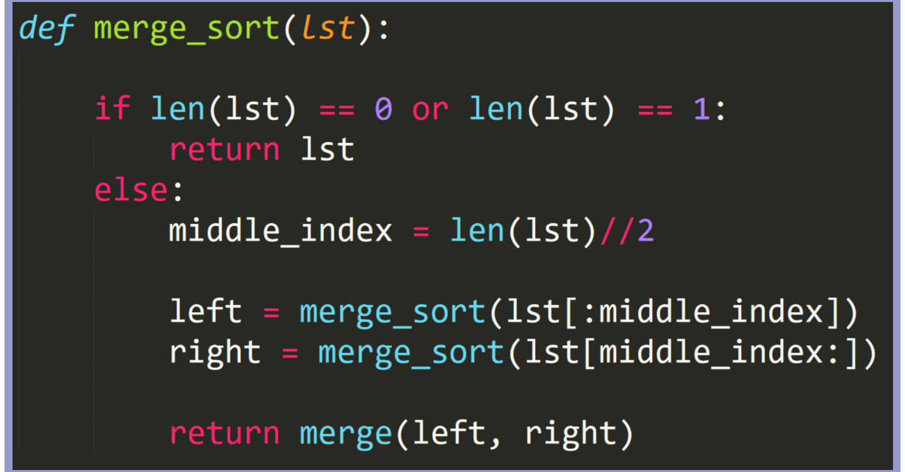
```python
def merge_sort(lst):

    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)
```



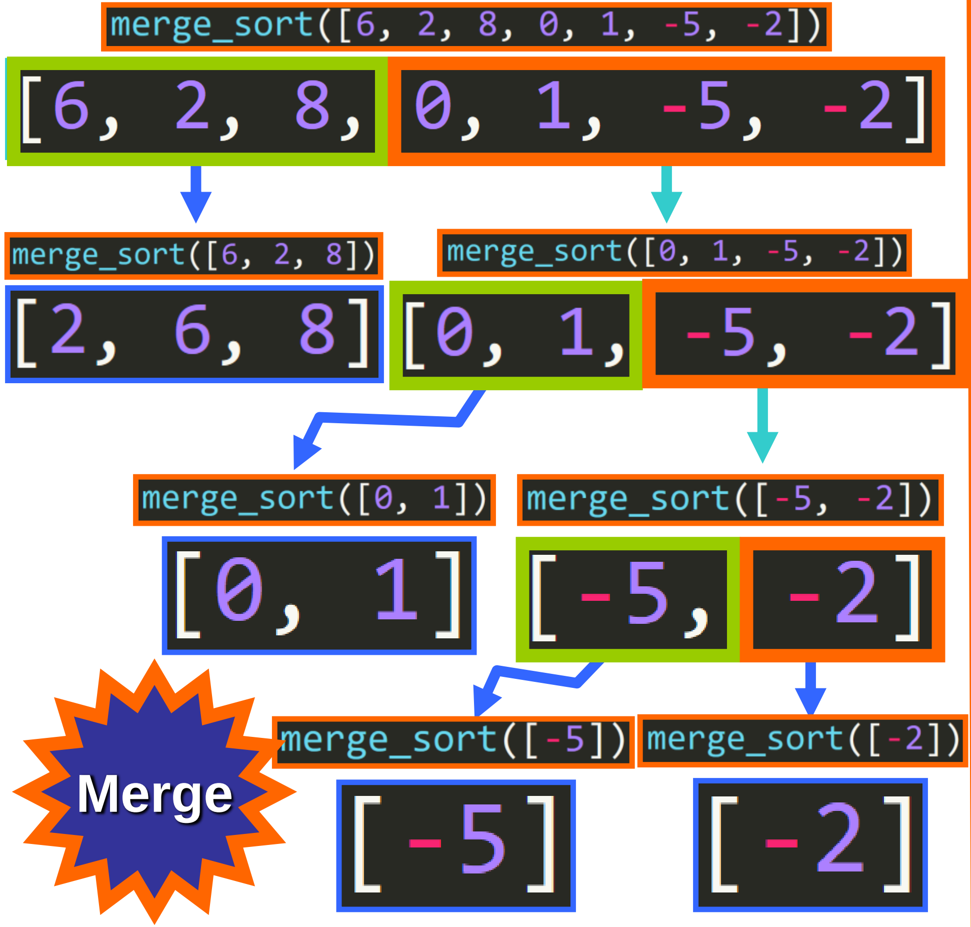


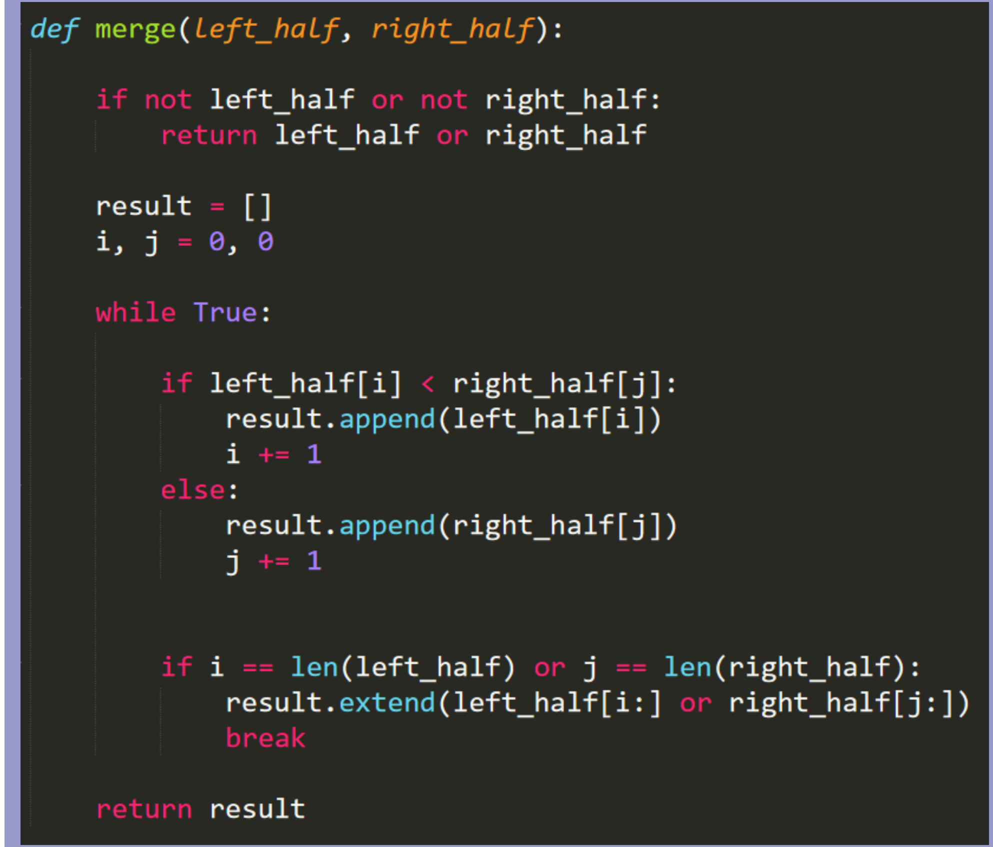
```
def merge_sort(lst):

    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)
```



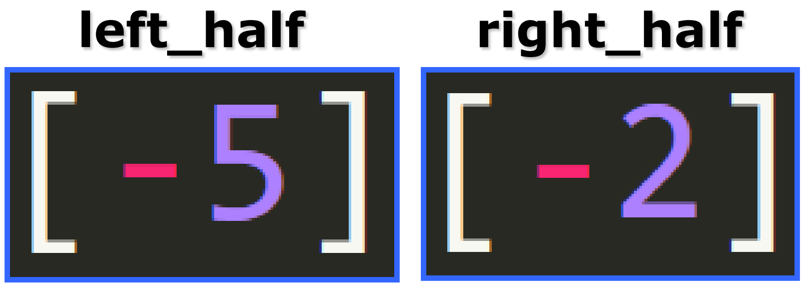


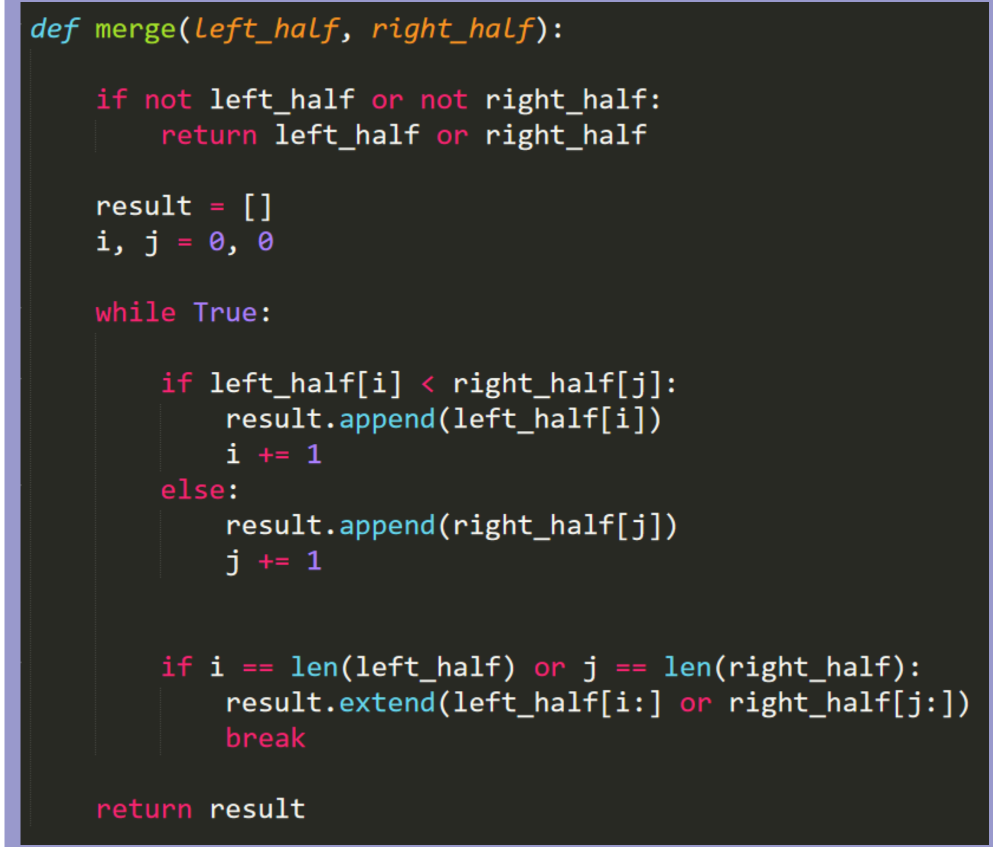
```python
def merge_sort(lst):

    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)
```





```python
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]:
            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
```



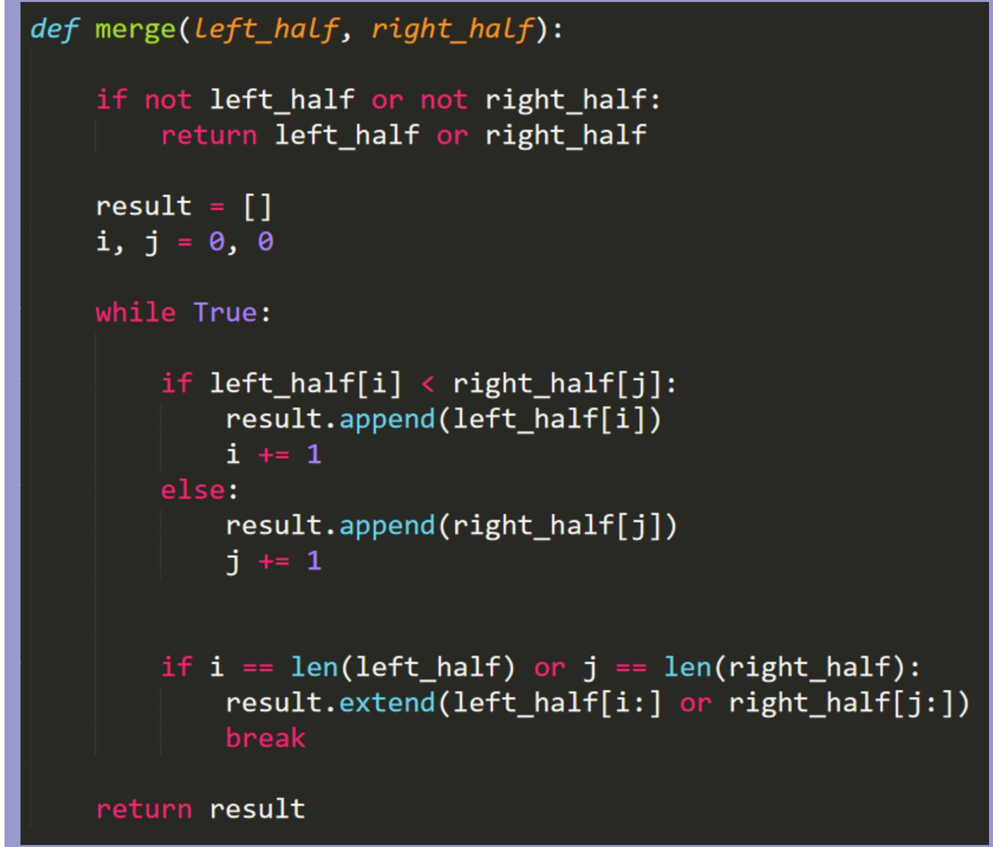


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0 j = 0

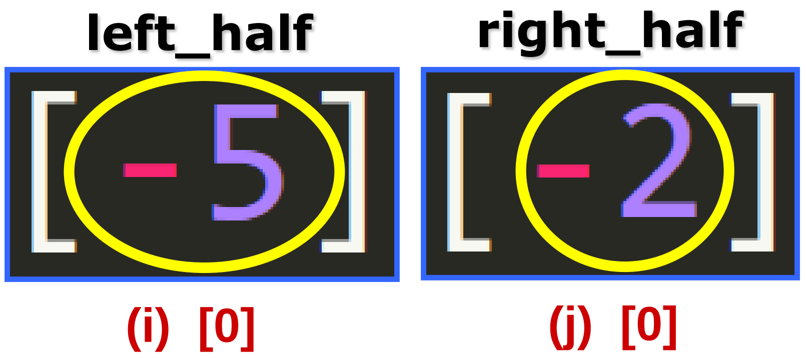
left_half	right_half
[-5]	[-2]
(i) [0]	(j) [0]

result = [ ]



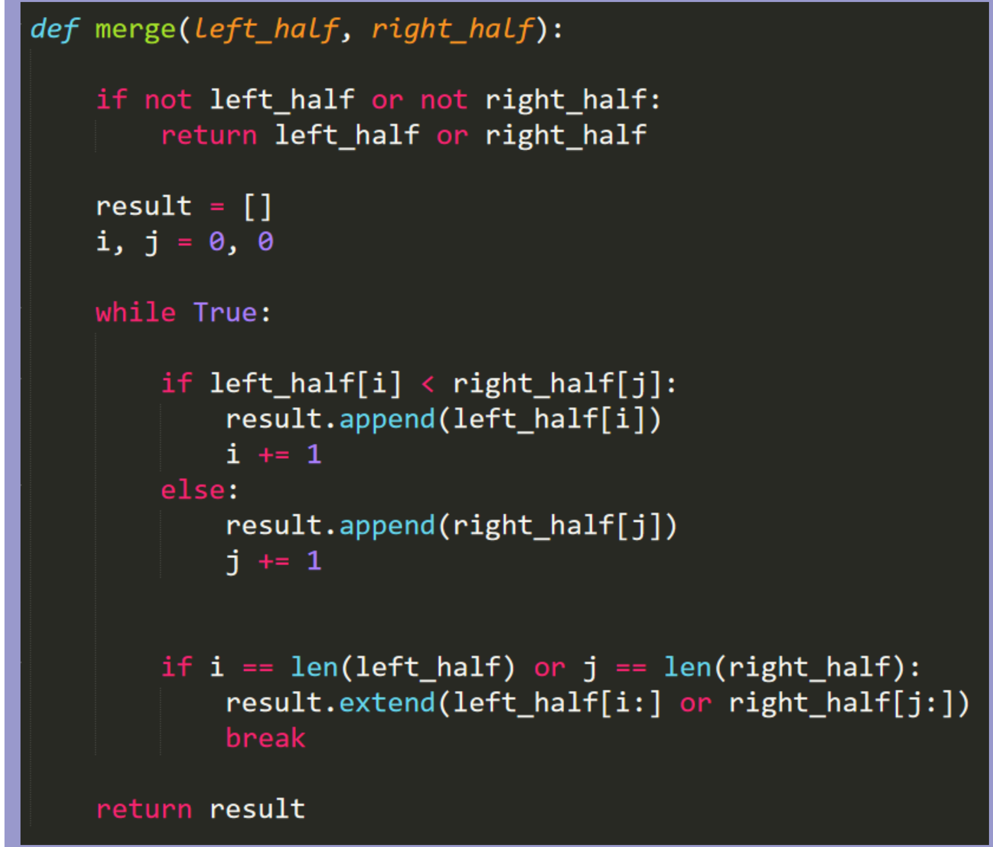
```
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 0

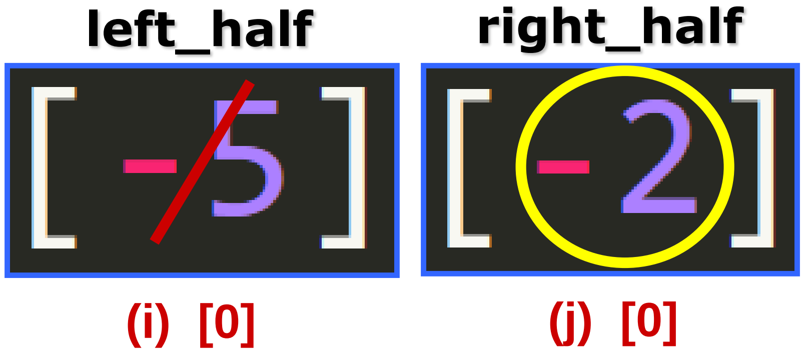


-5 < -2?

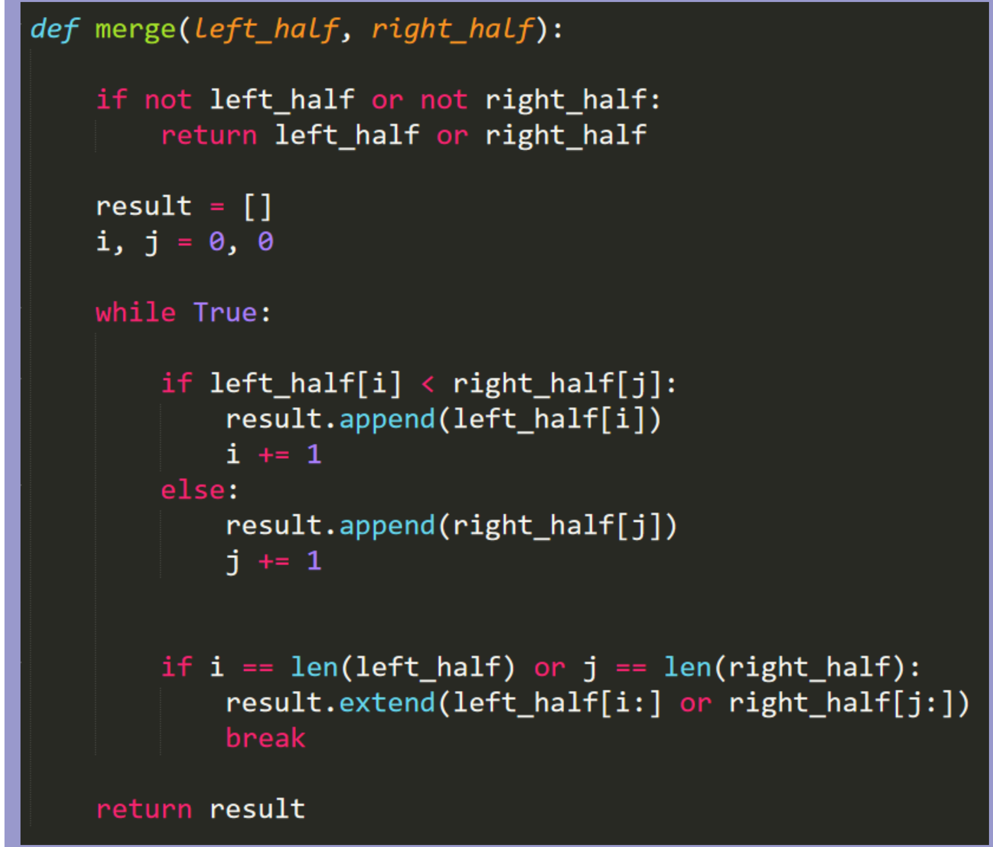
result = [ ]


```python
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]:
            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
```

i = 1   j = 0



result = [-5]

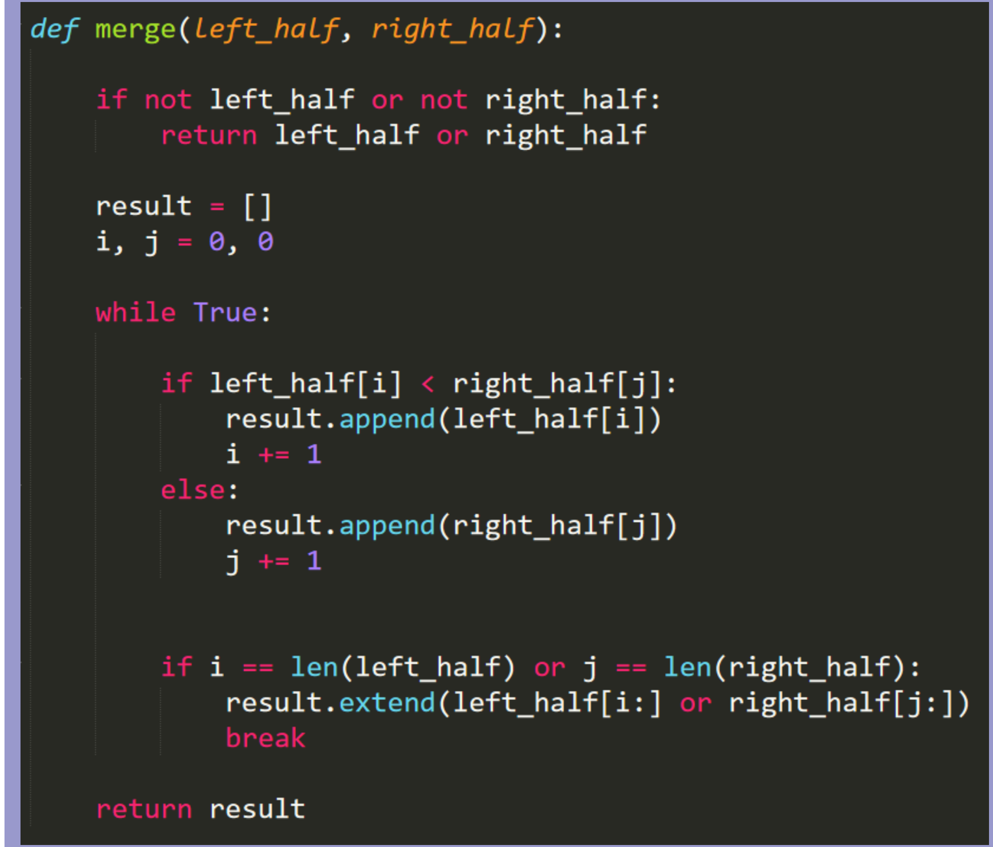


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 1   j = 0

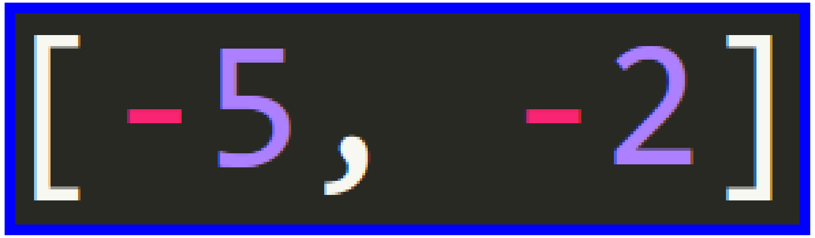
left_half	right_half
[~~-5~~]	[~~-2~~]
(i) [0]	(j) [0]

result = [-5, -2]



```python
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]:
            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
```

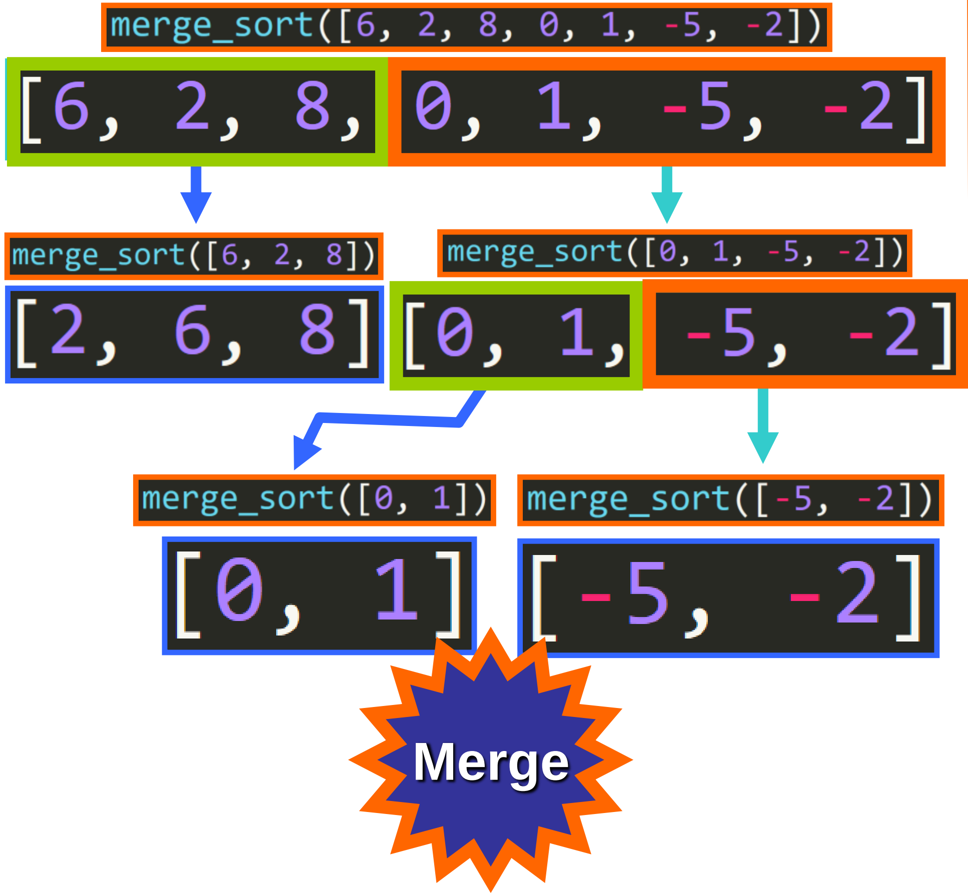
i = 1   j = 0

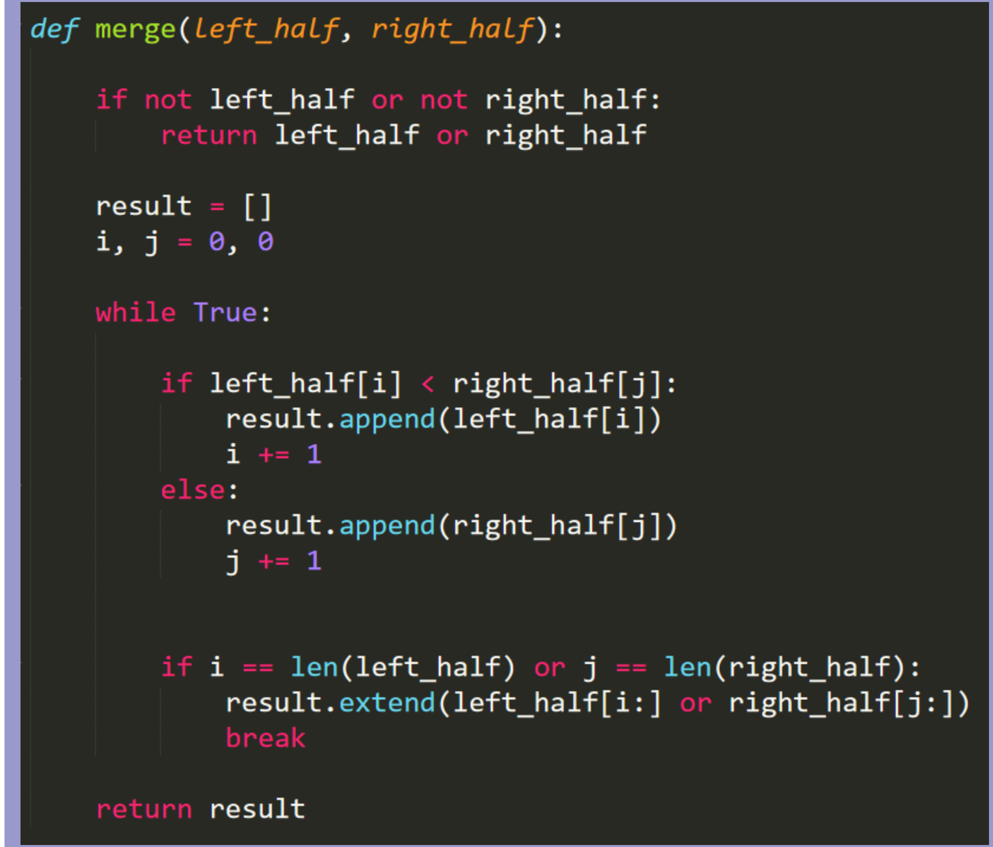


```
[-5, -2]
```

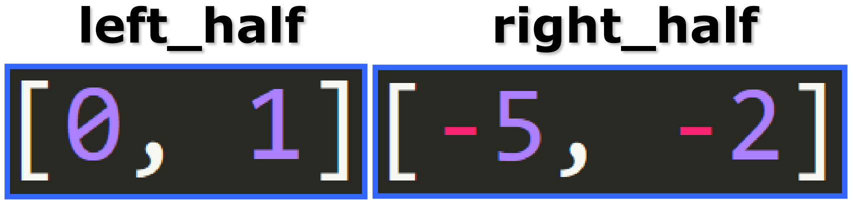
result = [-5, -2]

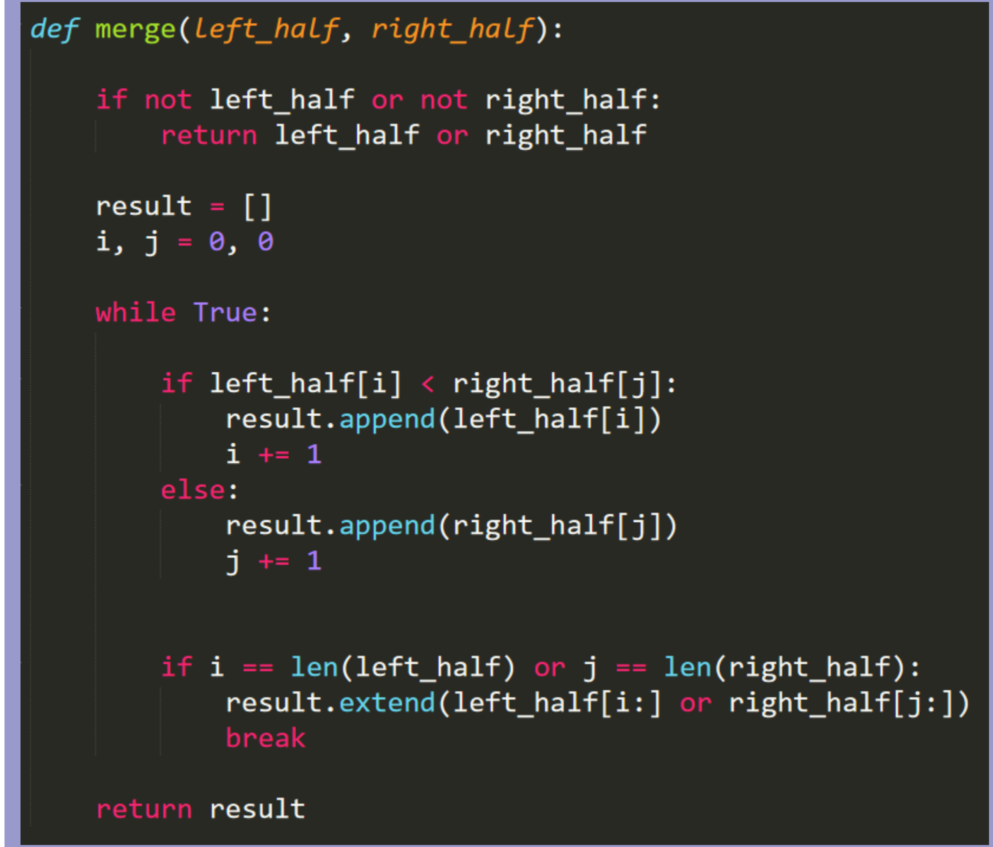
```python
def merge_sort(lst):

    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)
```



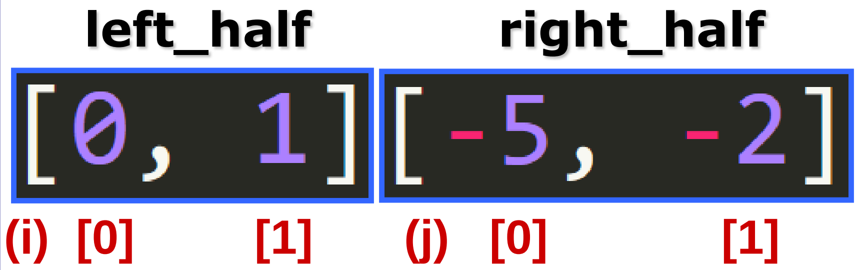


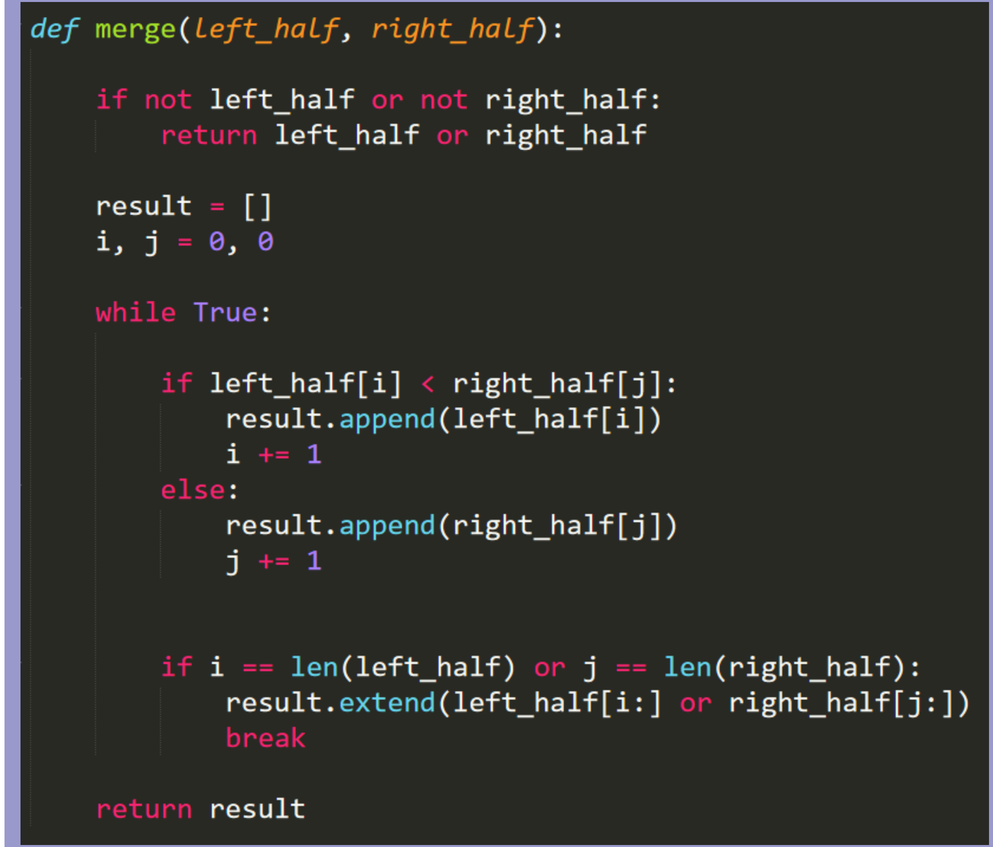
```python
def merge(left_half, right_half):

    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]:
            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
```





```python
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]:
            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
```





```python
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]:
            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
```

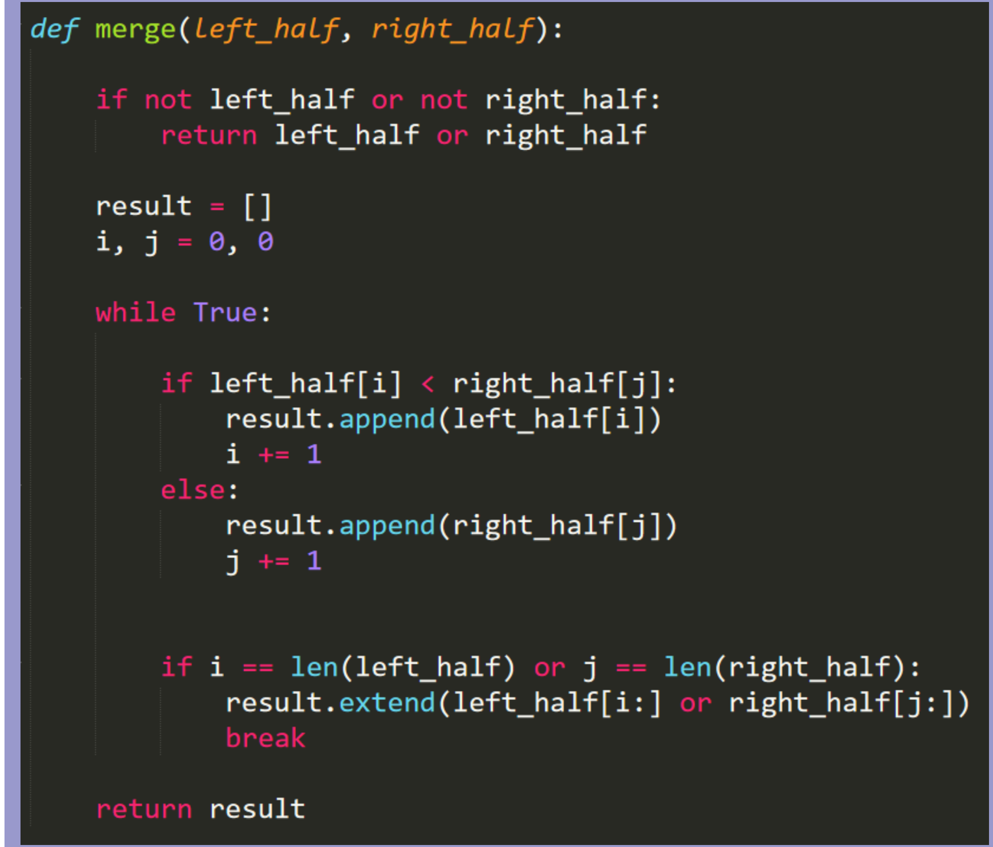
i = 0 j = 0

left_half **right_half**

[0, 1] [-5, -2]

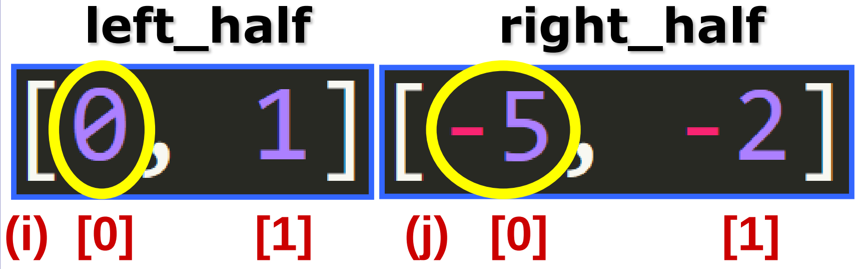
(i) [0] [1] (j) [0] [1]

result = [ ]



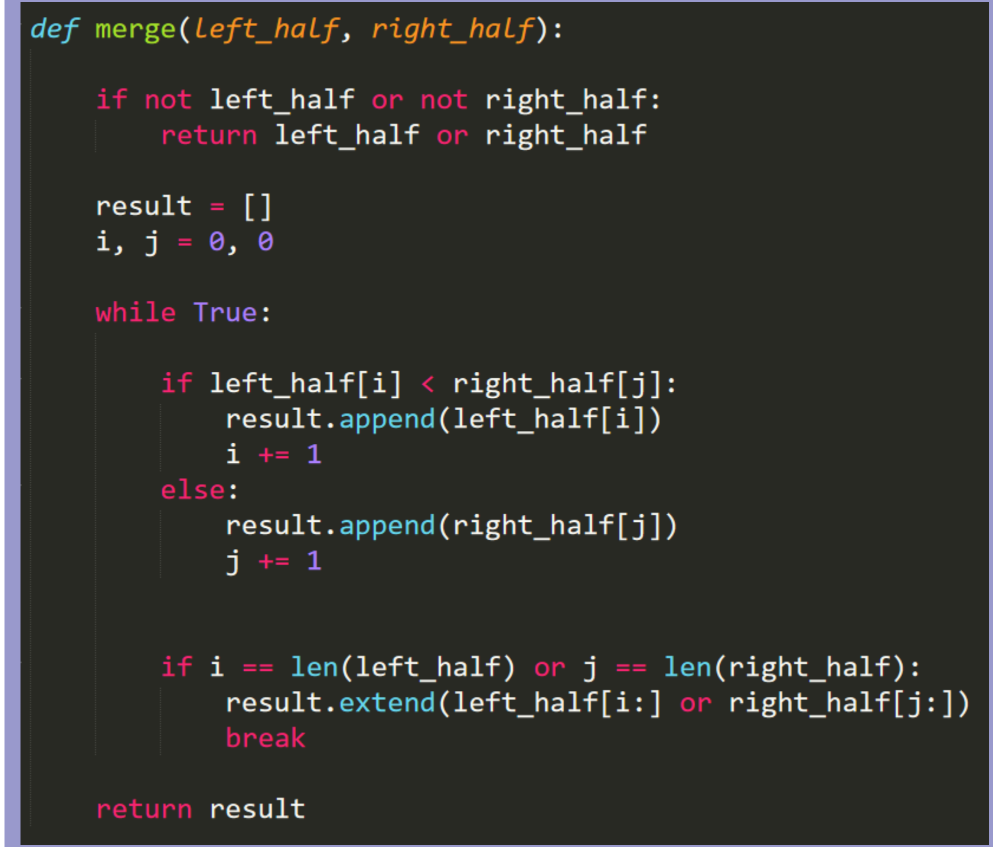
```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0 j = 0



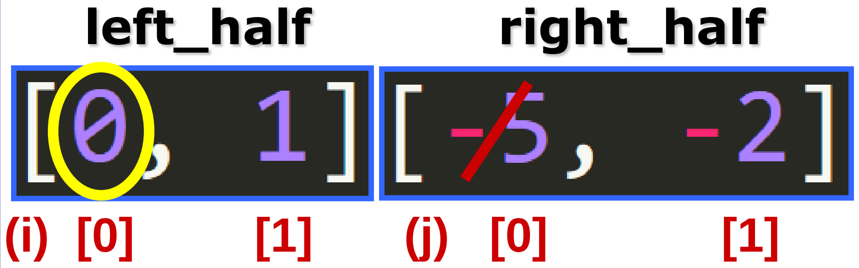
0 < -5?

result = [ ]

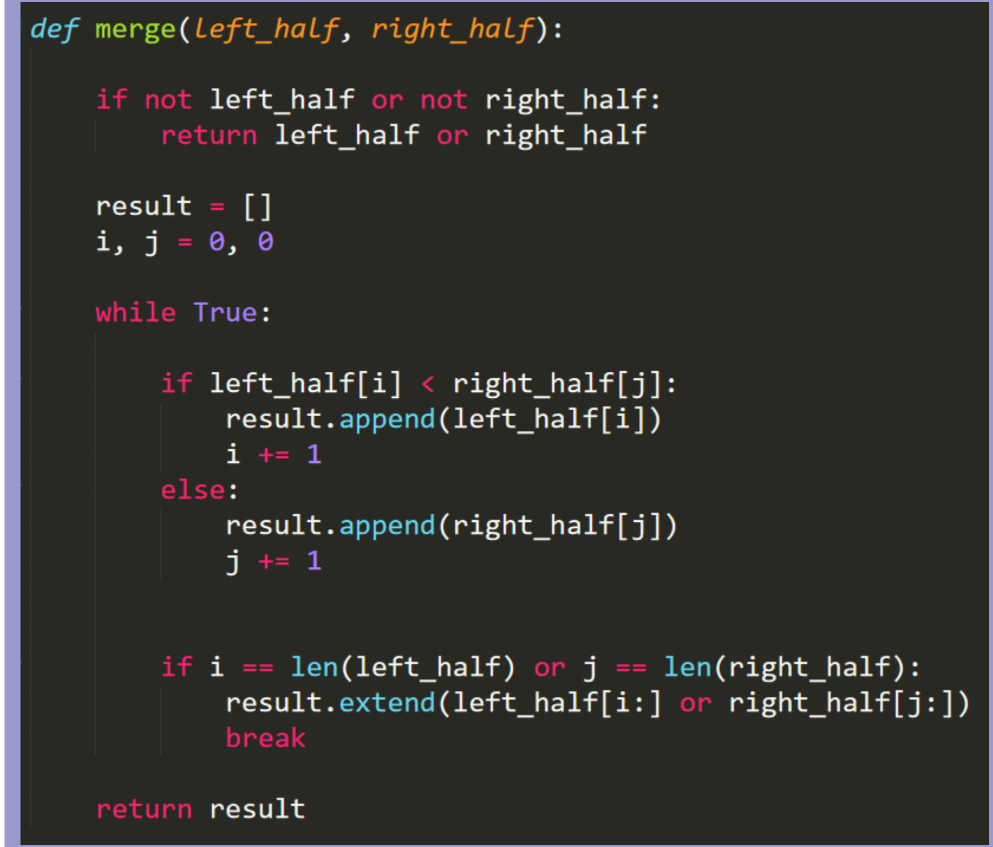


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 1

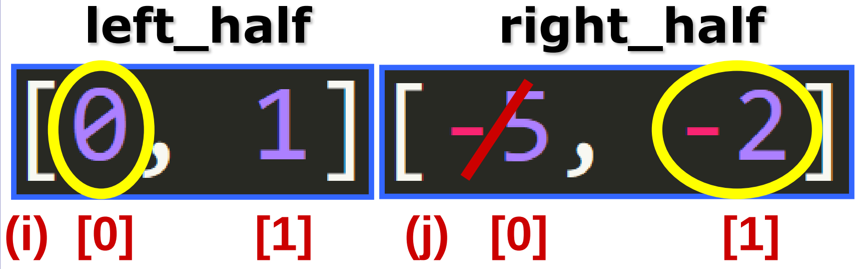


result = [-5]



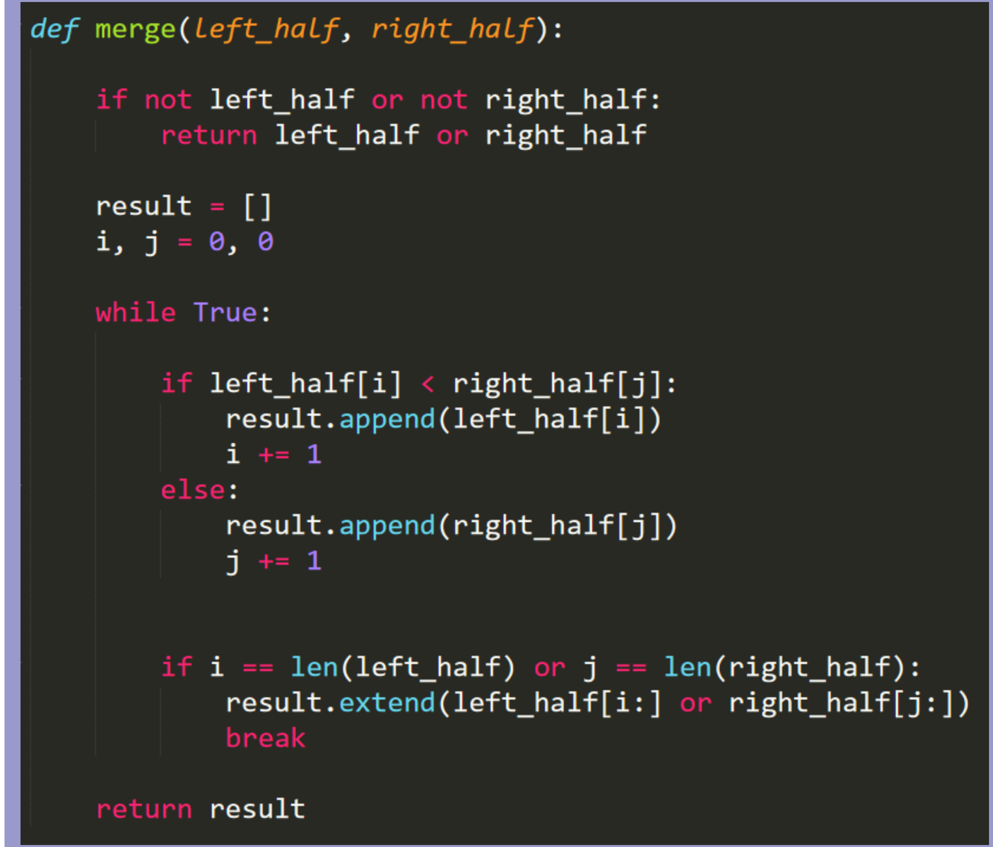
```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0 j = 1



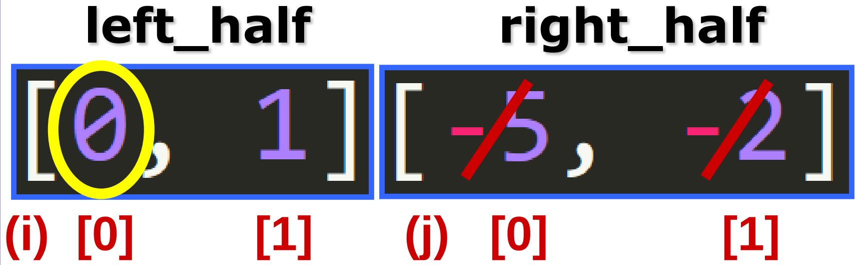
0 < -2?

result = [-5]

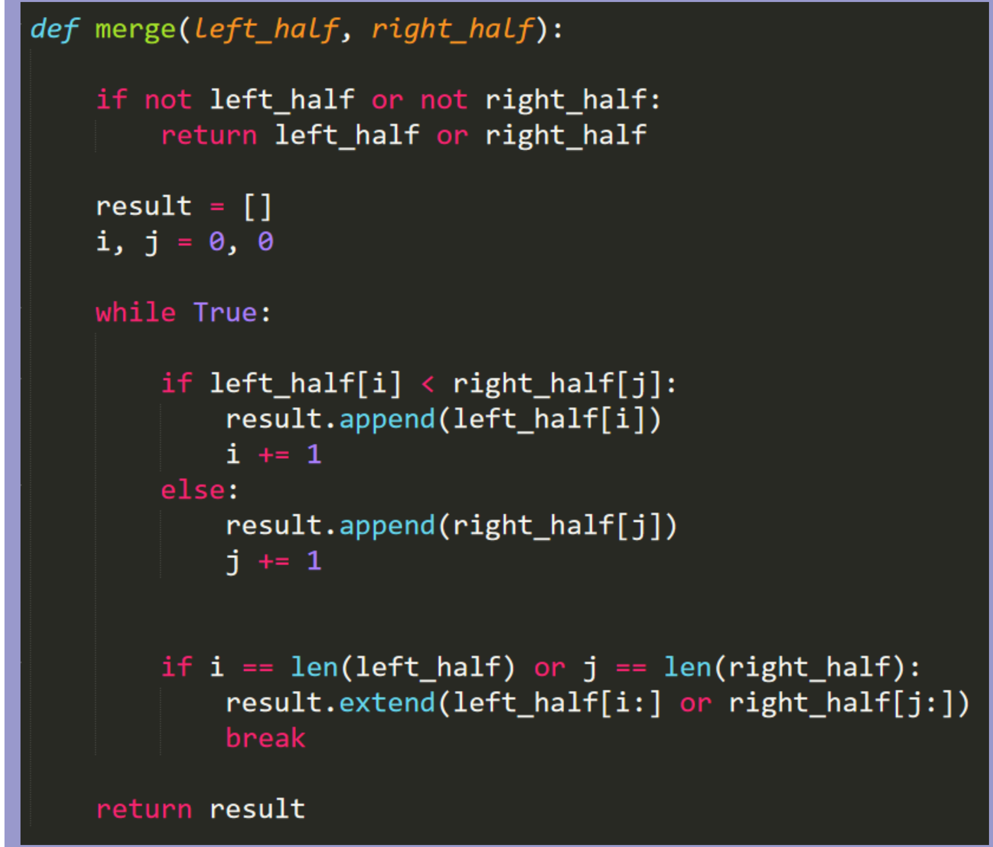


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0 j = 2



result = [-5, -2]

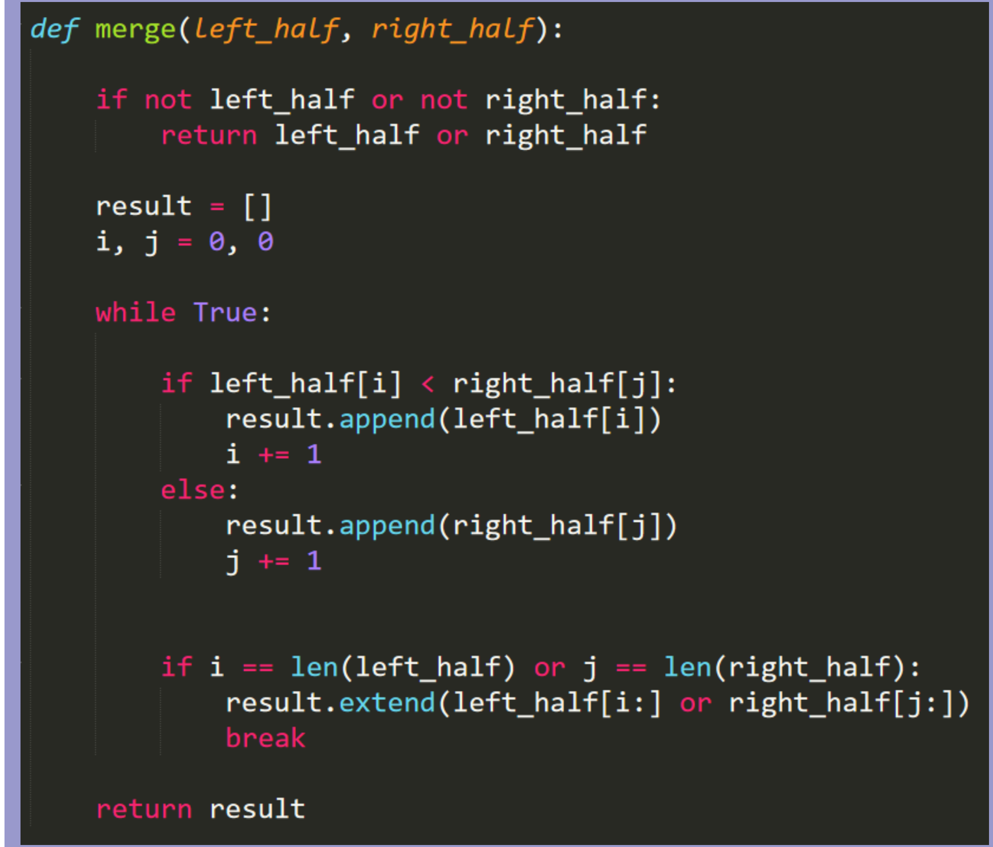


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 2

left_half		right_half	
0	1	~~-5~~	~~-2~~
(i) [0]	[1]	(j) [0]	[1]

result = [-5, -2]

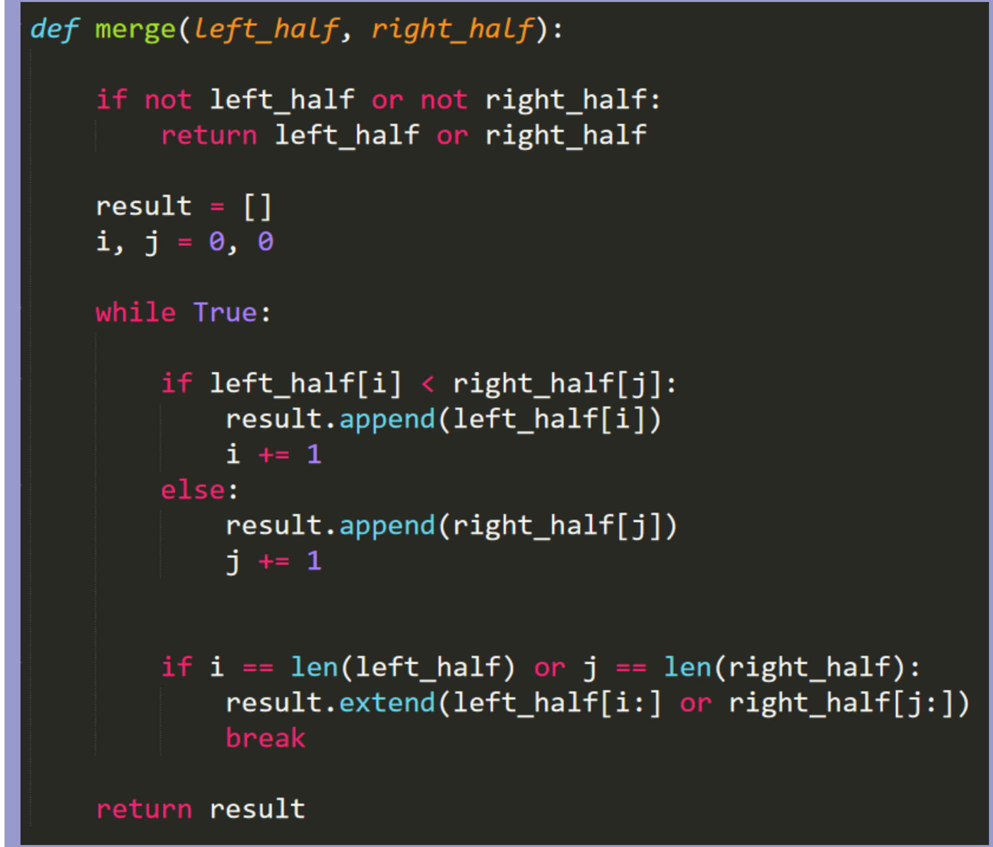


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 2

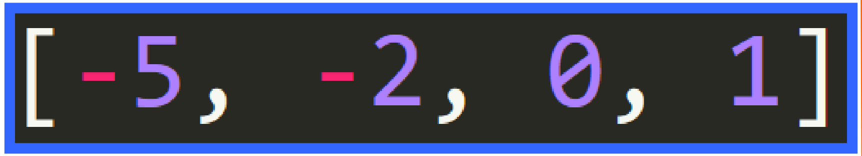
left_half		right_half	
~~0~~	~~1~~	~~-5~~	~~-2~~
(i) [0]	[1]	(j) [0]	[1]

result = [-5, -2, 0, 1]



```python
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]:
            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
```

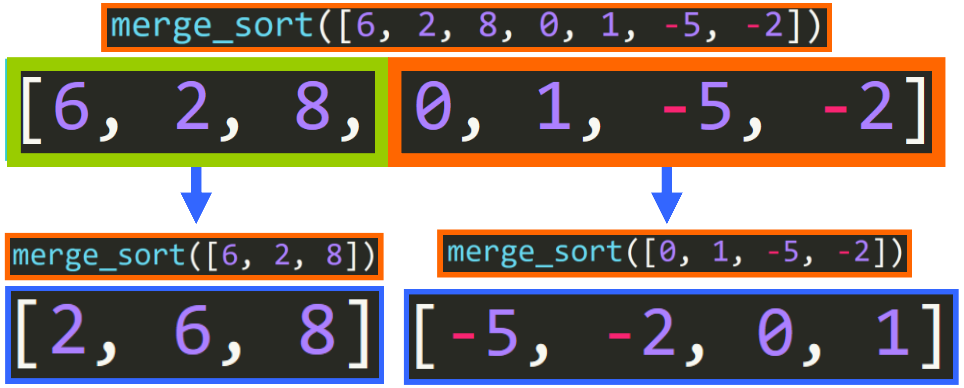
i = 0   j = 2



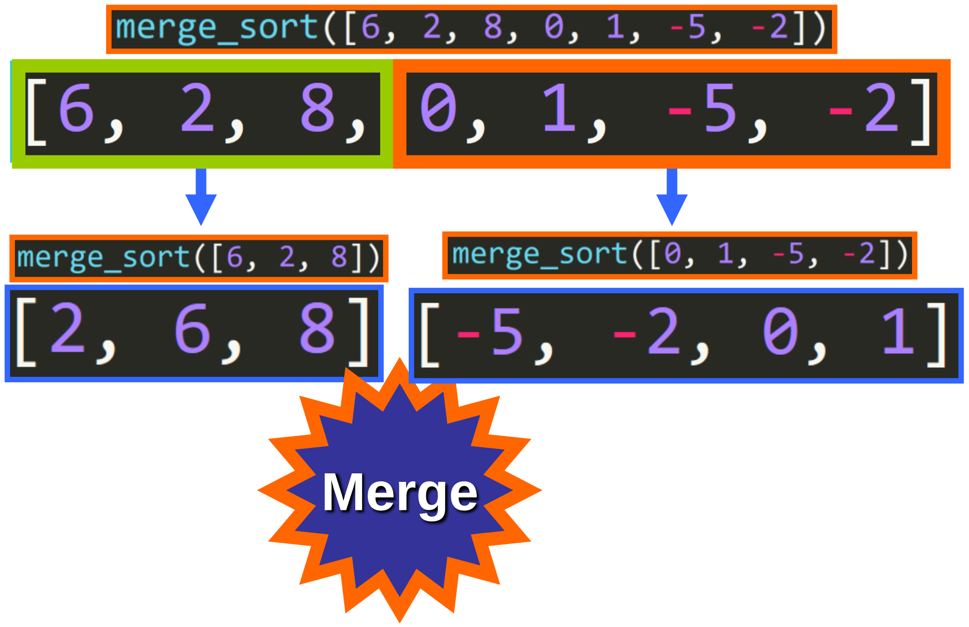
```
[-5, -2, 0, 1]
```

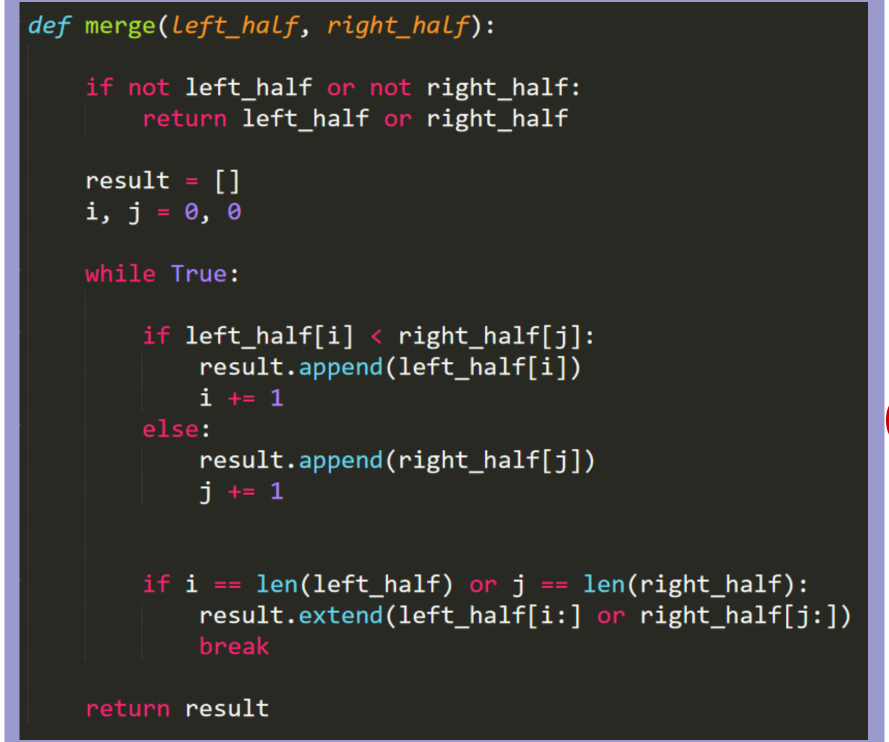
result = [-5, -2, 0, 1]

```python
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)
```

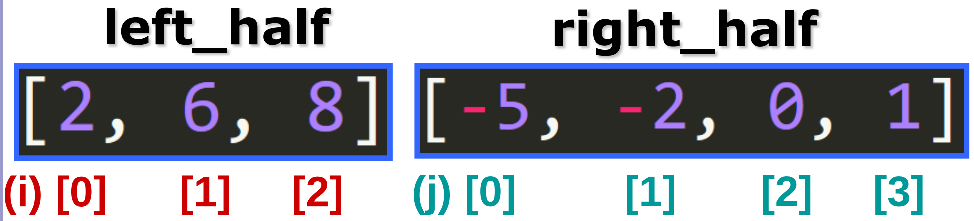


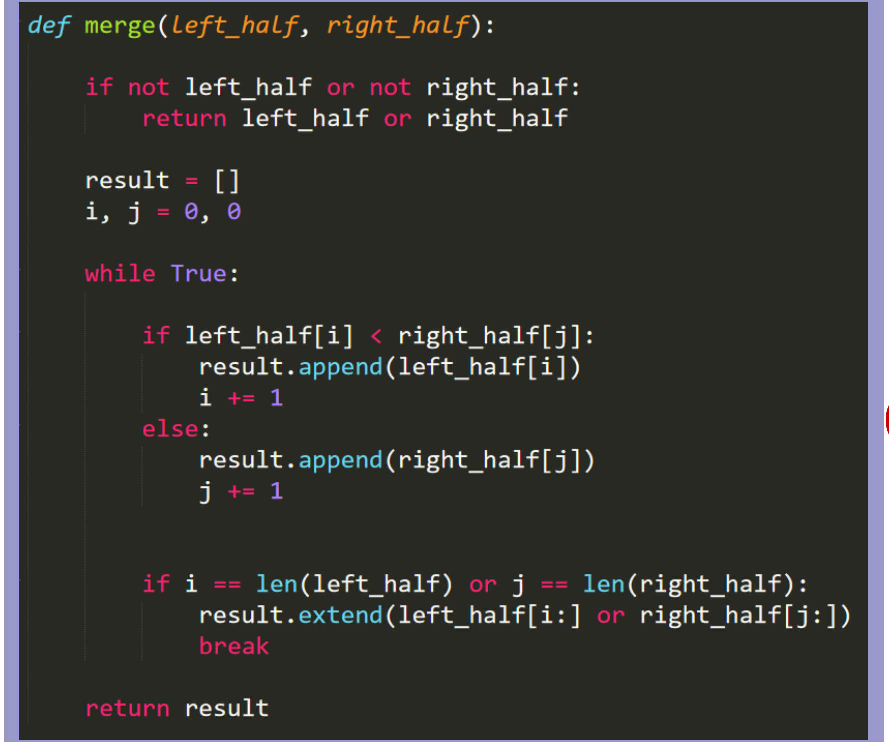
```python
def merge_sort(lst):

    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]:
            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
```

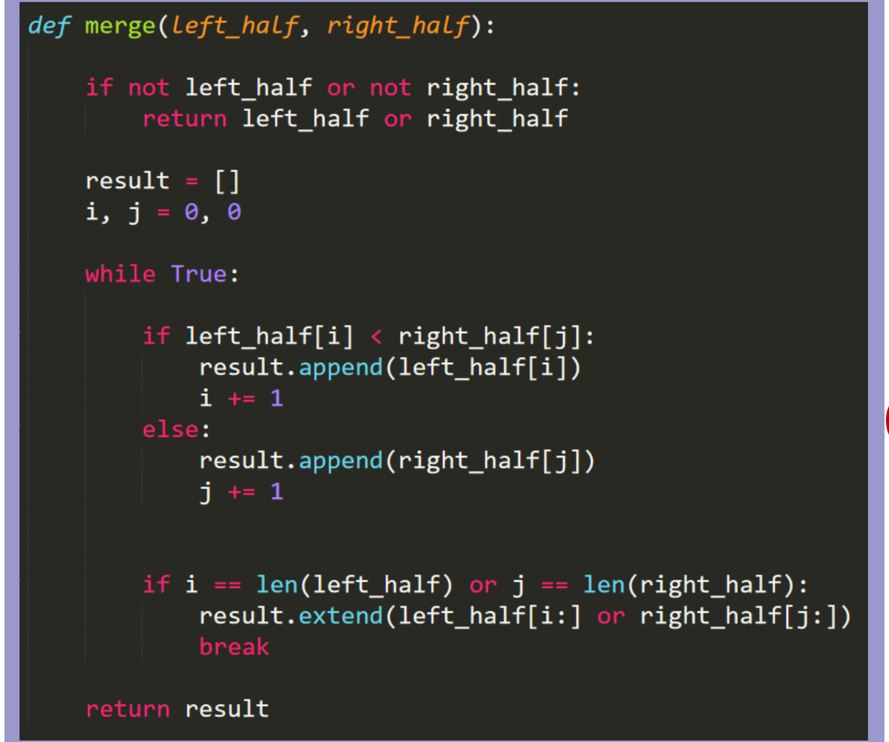




```python
def merge(left_half, right_half):

    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]:
            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
```

left_half **right_half**

```
[2, 6, 8]            [-5, -2, 0, 1]
(i) [0]  [1]  [2]    (j) [0]  [1]  [2]  [3]
```



```python
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]:
            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
```

i = 0   j = 0

left_half

[2, 6, 8]

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

right_half

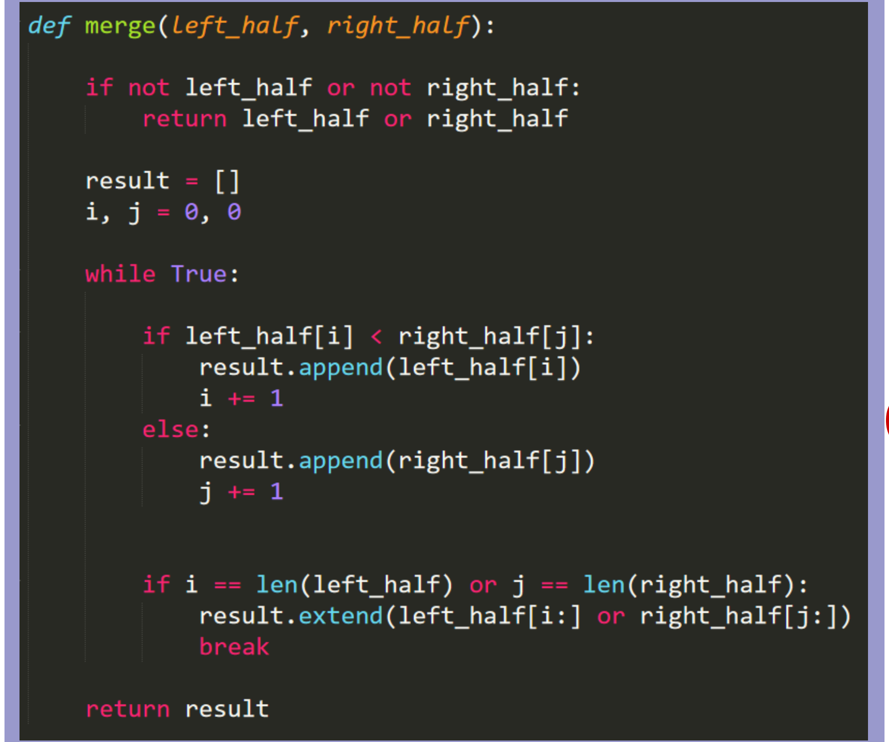
[-5, -2, 0, 1]

(j) [0] [1] [2] [3]

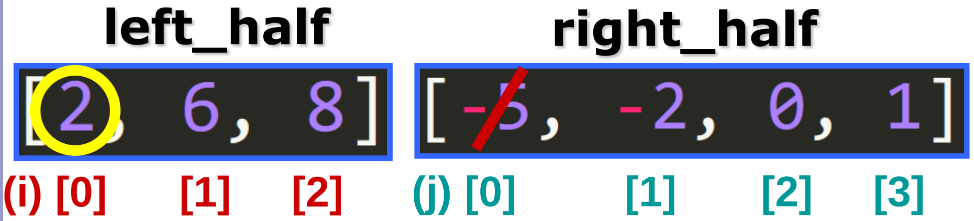
2 < -5?

result = [ ]

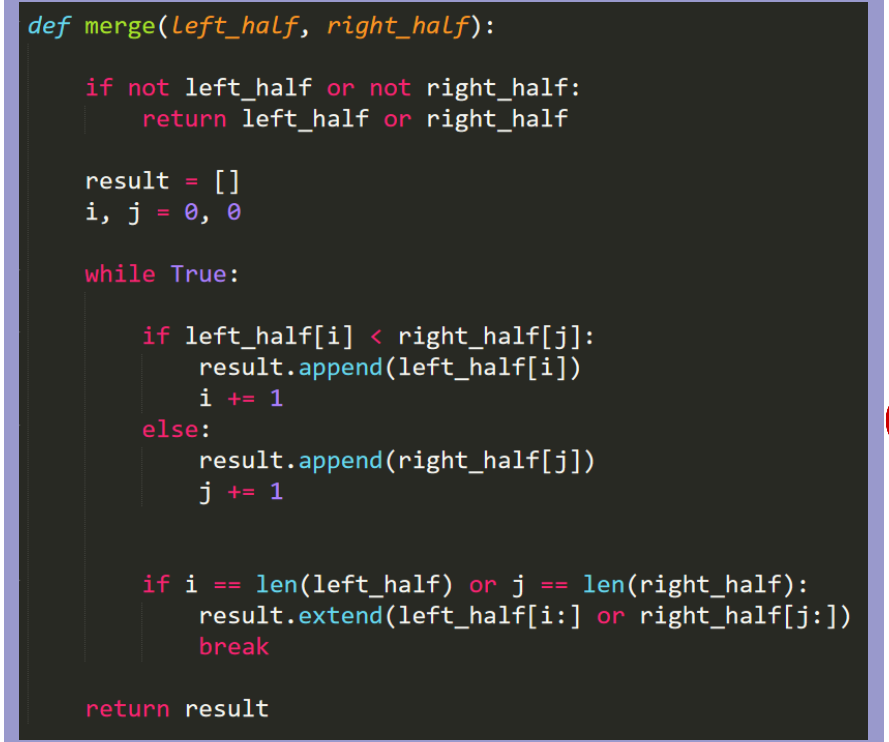
i = 0   j = 1



```python
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]:
            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
```



result = [-5]



```python
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]:
            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
```

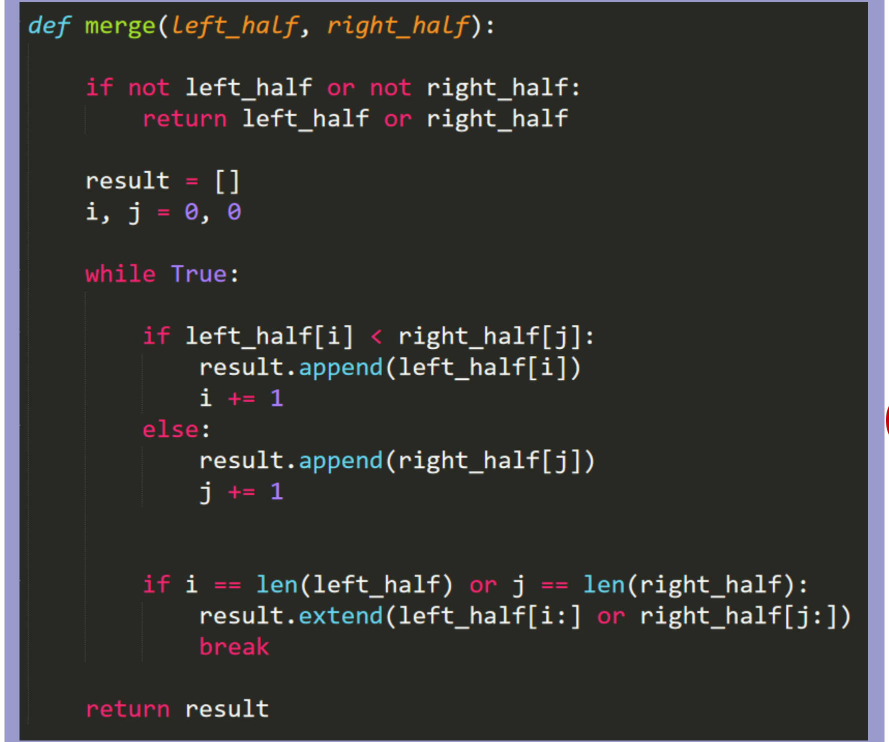
i = 0 j = 1

left_half **right_half**

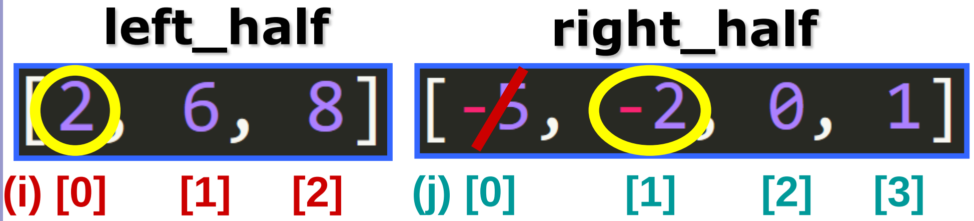
[2, 6, 8] [~~-5~~, -2, 0, 1]

(i) [0] [1] [2] (j) [0] [1] [2] [3]

result = [-5]

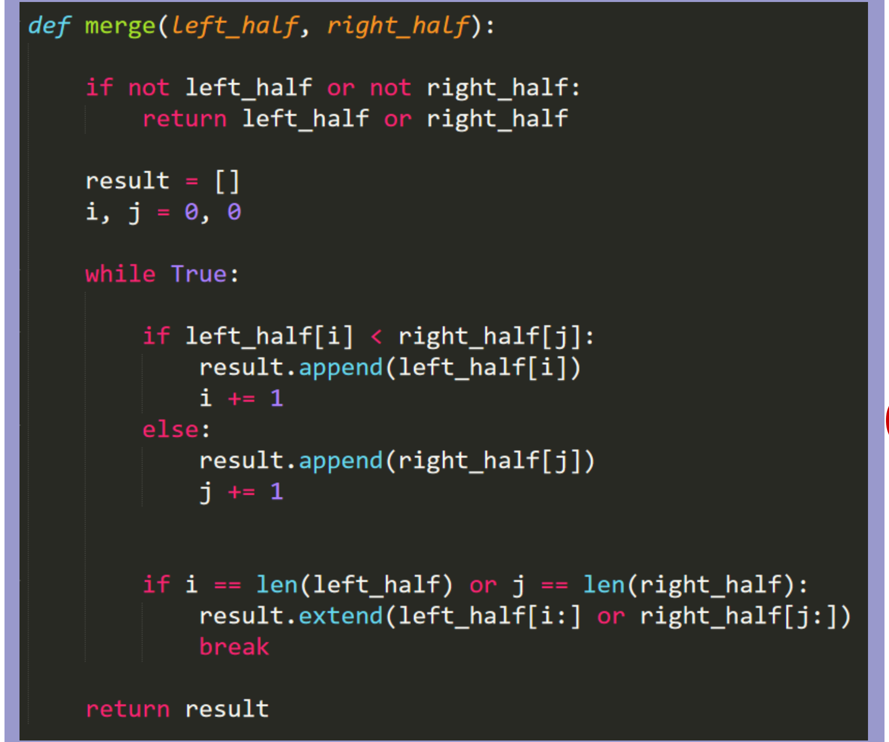

```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 1



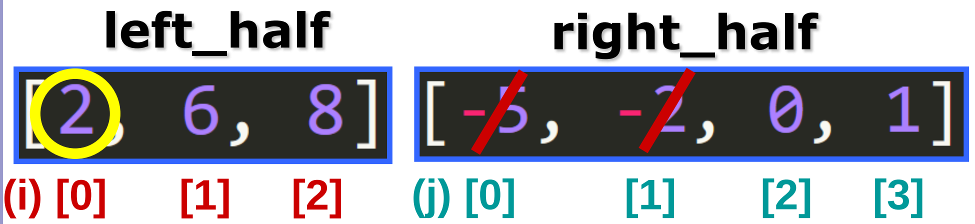
2 < -2?

result = [-5]

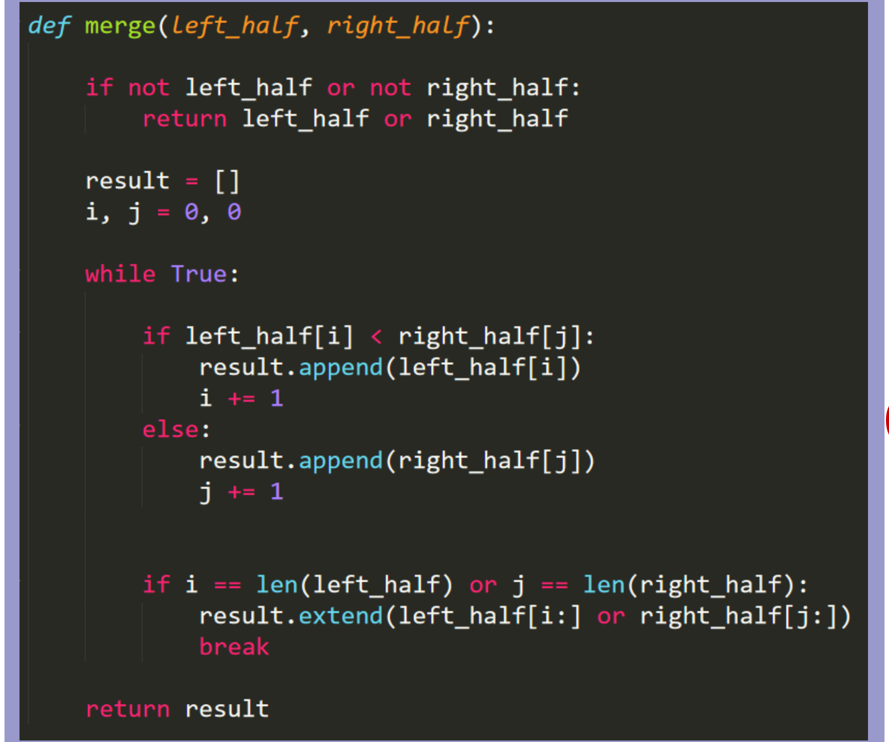


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 2

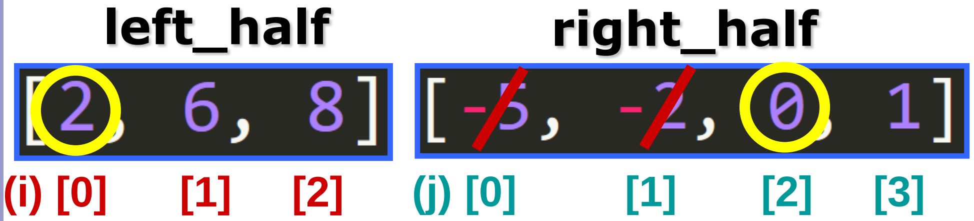


result = [-5, -2]



```python
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]:
            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
```

i = 0   j = 2

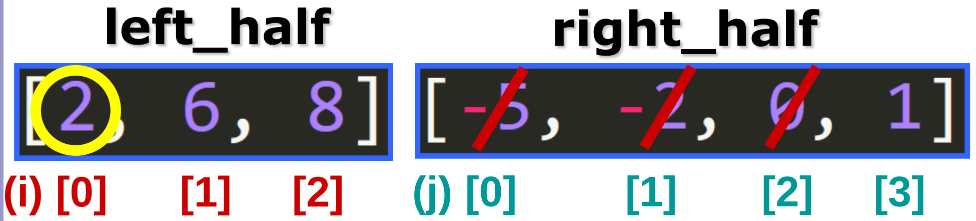


2 < 0?

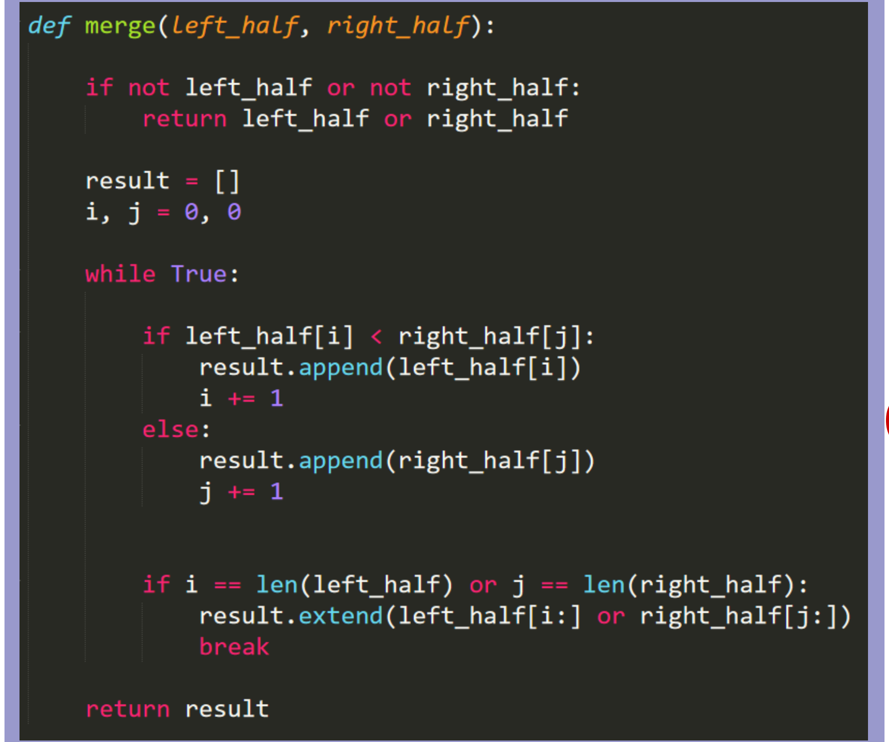
result = [-5, -2]

```
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]:
            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
```

i = 0   j = 3

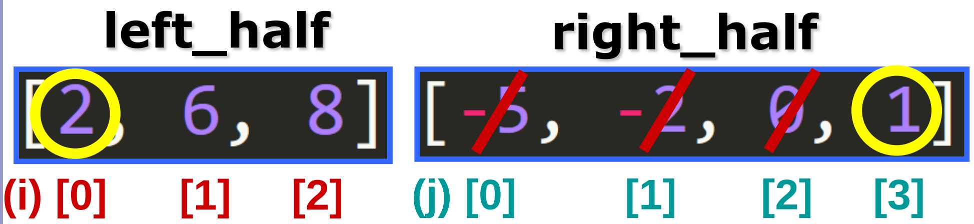


result = [-5, -2, 0]



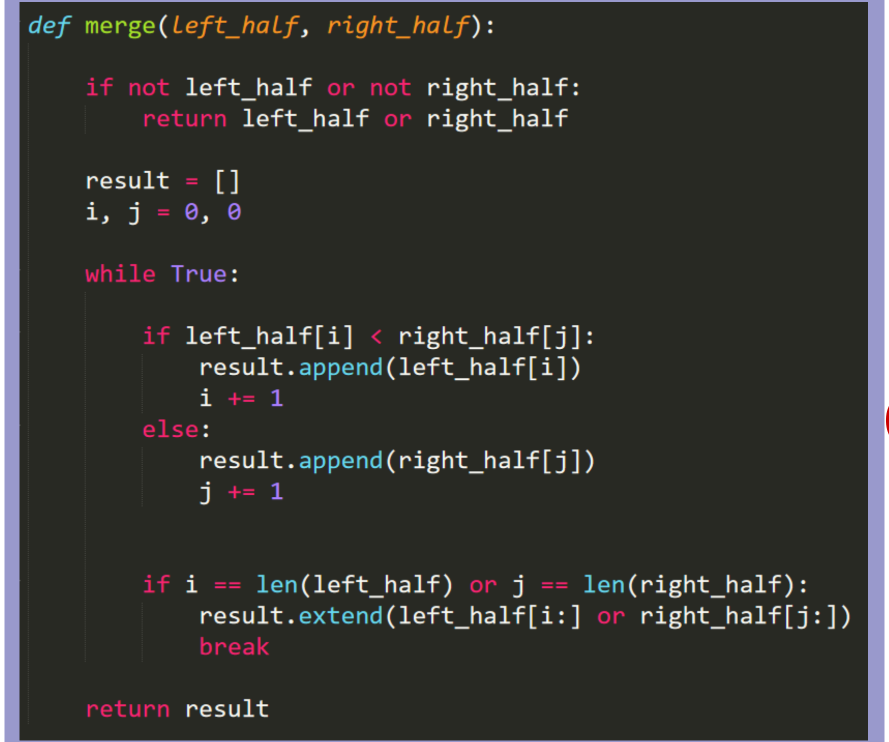
```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0 j = 3



2 < 1?

result = [-5, -2, 0]

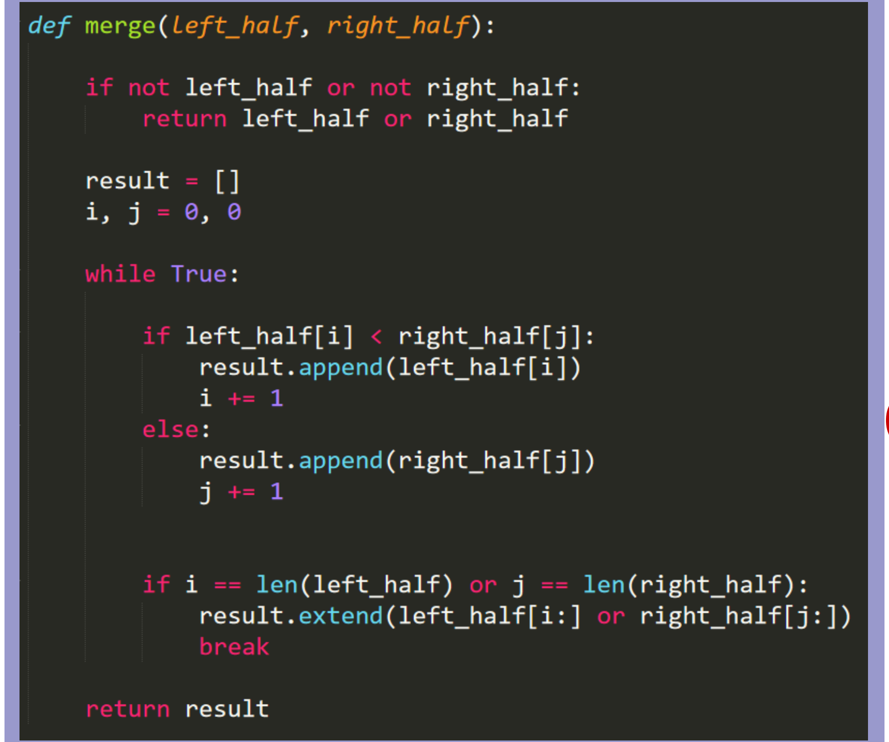


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 4

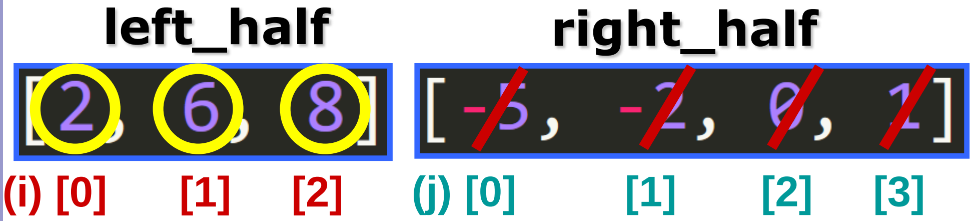
left_half			right_half			
[2,	6,	8]	[$\cancel{-5}$,	$\cancel{-2}$,	$\cancel{0}$,	$\cancel{1}$]
(i) [0]	[1]	[2]	(j) [0]	[1]	[2]	[3]

result = [-5, -2, 0, 1]

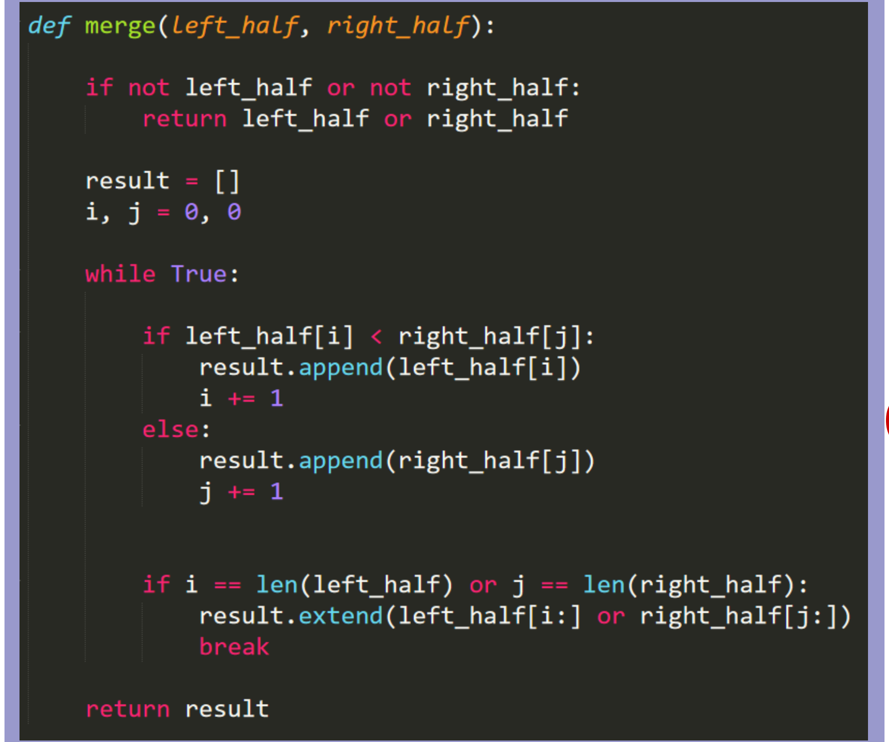


```
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 4

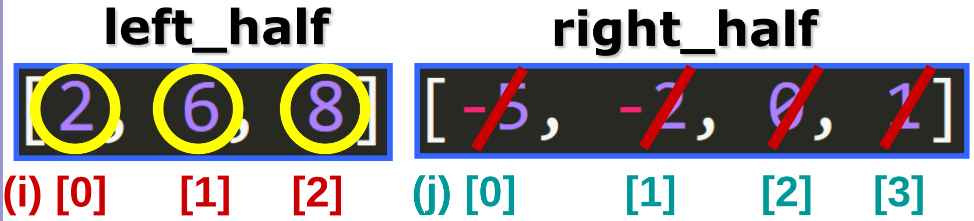


result = [-5, -2, 0, 1]

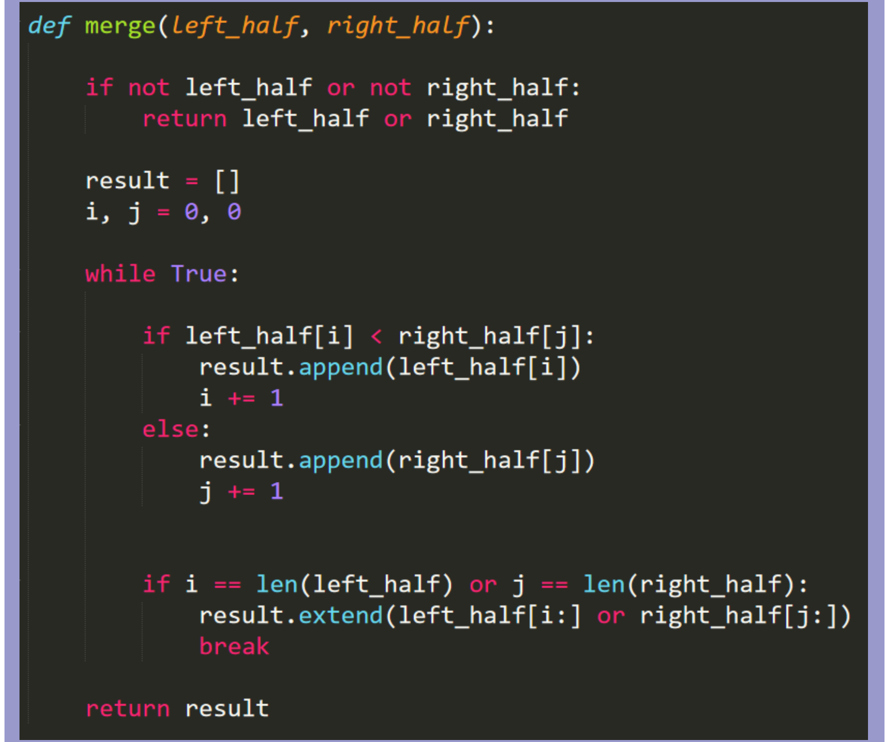


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 4



result = [-5, -2, 0, 1, 2, 6, 8]

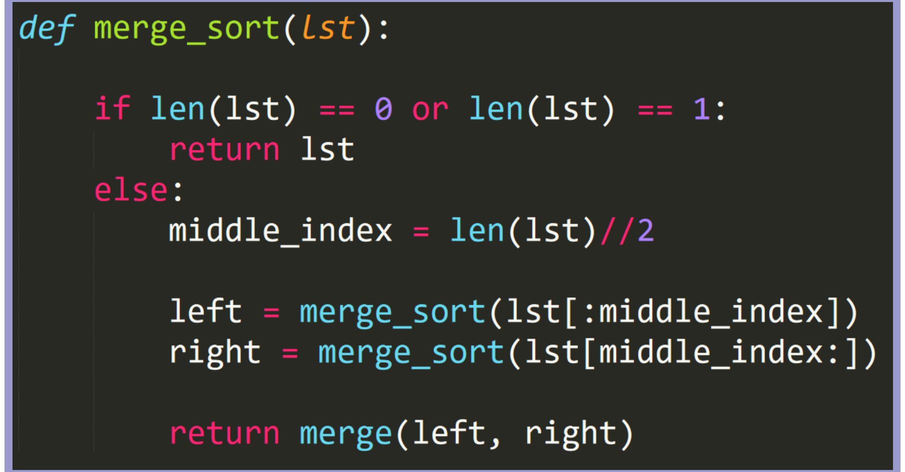


```python
def merge(left_half, right_half):

    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]:
            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
```

i = 0   j = 4

```
[-5, -2, 0, 1, 2, 6, 8]
```

result = [-5, -2, 0, 1, 2, 6, 8]



```python
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([6, 2, 8, 0, 1, -5, -2])
```

```
[-5, -2, 0, 1, 2, 6, 8]
```

Sorted!

Time to Practice