```
# 두 리스트 합치기
def merge(list1, list2):
    # 코드를 입력하세요.

# 합병 정렬
def merge_sort(my_list):
    # 코드를 입력하세요.

some_list = [11, 3, 6, 4, 12, 1, 2]
sorted_list = merge_sort(some_list)
print(sorted_list)
```

합병 정렬(Merge Sort)은 재귀함수가 포함된 정렬 알고리즘으로, 선택 정렬이나 삽입 정렬보다 효율적입니다.

merge 함수

먼저 **merge** 함수에 대해 살펴봅시다. **merge** 함수는 정렬되어 있는 **list1** 과 **list2** 라는 두 리스트를 파라미터로 받습니다.

두 리스트는 이미 정렬되어있기에, 합쳐진 리스트(**merged_list**)에 추가될 원소는 두 리스트의 왼쪽 값 중 더 작은 값입니다.

인덱스 **i** 가 **len(list1)** 보다 작고, 인덱스 **j** 가 **len(list2)** 보다 작을 동안, **list1** 과 **list2** 의 원소들을 차례로 비교하여, 더 작은 원소를 **merged_list** 에 추가합니다.

예를 들어 **list1** 이 **[2, 4]** 이고, **list2** 가 **[1, 3, 5, 6]** 라고 가정합시다.

실행 순서

1. **i = 0** , **j = 0** 일 때: **list1[0]** 이 **2** , **list2[0]** 이 **1** 임. **2 > 1** 이므로 **list2[0]** 이 **merged_list** 에 추가되고, **j** 의 값이 **1** 증가함.
2. **i = 0** , **j = 1** 일 때: **list1[0]** 이 **2** , **list2[1]** 이 **3** 임. **2 < 3** 이므로 **list1[0]** 이 **merged_list** 에 추가되고, **i** 의 값이 **1** 증가함.
3. **i = 1** , **j = 1** 일 때: **list1[1]** 이 **4** , **list2[1]** 이 **3** 임. **4 > 3** 이므로 **list2[1]** 이 **merged_list** 에 추가되고, **j** 의 값이 **1** 증가함.
4. **i = 1** , **j = 2** 일 때: **list1[1]** 이 **4** , **list2[2]** 이 **5** 임. **4 < 5** 이므로 **list1[1]** 이 **merged_list** 에 추가되고, **i** 의 값이 **1** 증가함.
5. **i** 의 값이 **2** , 즉 **len(list1)** 과 같아지므로, 반복문이 종료됨.

위의 과정에서 **list1** 과 **list2** 의 원소들을 차례로 비교하여 더해주었는데, 아직 **list2** 에 있는 **5** 와 **6** 이라는 원소가 더해지지 않았습니다! **list1** 이나 **list2** 에 남은 원소가 있다면, **merged_list** 에 추가해주어야 합니다.

이 내용을 코드로 구현한 결과는 아래와 같습니다:

```
# 두 리스트 합치기
def merge(list1, list2):
    # initialize
    merged_list = []
    i = 0
    j = 0

    # 정렬되어 있는 list1과 list2의 원소들을 차례로 비교하여, 더 작은 원소를 merged_lis
t에 추가하기
    while i < len(list1) and j < len(list2):
        if list1[i] <= list2[j]:
            merged_list.append(list1[i])
            i += 1
        else:
            merged_list.append(list2[j])
            j += 1

    # list1에 남은 원소가 있다면, merged_list에 추가하기
    if i < len(list1):
        while i < len(list1):
            merged_list.append(list1[i])
            i += 1

    # list2에 남은 원소가 있다면, merged_list에 추가하기
    if j < len(list2):
        while j < len(list2):
            merged_list.append(list2[j])
            j += 1

    return merged_list
```

merge_sort 함수

이제 분할 - 정복 개념을 사용하여, **merge_sort** 함수를 써야 합니다.

Base Case

우선 Base case를 생각해봅시다. 파라미터인 **my_list** 의 길이가 **0** 또는 **1** 이라면, 그 자체가 정렬된 리스트이겠죠?

if len(my_list) <= 1: 인 경우에는 **my_list** 그 자체를 리턴해주면 됩니다.

Recursive Case

Recursive case를 생각해봅시다. 우리가 **merge_sort** 함수를 올바르게 짰다고 가정합시다.

그렇다면 파라미터인 **my_list** 를 반으로 쪼개서 **left** 와 **right** 에 각각 지정한 후, **left** 와 **right** 을 **merge_sort** 의 파라미터로 넘겨주면 각자 정렬이 되어야겠죠? 이 정렬된 두 리스트를, 앞서 작성한

merge 함수에 넘겨주면 리스트가 정렬됩니다.

이 내용을 코드로 구현한 결과는 아래와 같습니다:

```
# 합병 정렬
def merge_sort(my_list):
    # Base Case
    if len(my_list) < 2:
        return my_list

    # Recursive Case: left와 right로 my_list를 나누어준다.
    left = my_list[:len(my_list) // 2]
    right = my_list[len(my_list) // 2:]

    # Recursive Case: my_list를 잘게 쪼개는 과정을 재귀적으로 반복하고, merge 함수를 사용하여 합쳐준다.
    return merge(merge_sort(left), merge_sort(right))

some_list = [11, 3, 6, 4, 12, 1, 2]
sorted_list = merge_sort(some_list)
print(sorted_list)
```

모범 답안

```
# 두 리스트 합치기
def merge(list1, list2):
    merged_list = []
    i = 0
    j = 0

    # 정렬되어 있는 list1과 list2의 원소들을 차례로 비교하여, 더 작은 원소를 merged_lis
t에 추가하기
    while i < len(list1) and j < len(list2):
        if list1[i] <= list2[j]:
            merged_list.append(list1[i])
            i += 1
        else:
            merged_list.append(list2[j])
            j += 1

    # list1에 남은 원소가 있다면, merged_list에 추가하기
    if i < len(list1):
        while i < len(list1):
            merged_list.append(list1[i])
            i += 1

    # list2에 남은 원소가 있다면, merged_list에 추가하기
    if j < len(list2):
        while j < len(list2):
            merged_list.append(list2[j])
            j += 1

    return merged_list

# 합병 정렬
def merge_sort(my_list):
    # Base Case
    if len(my_list) <= 1:
        return my_list

    # Recursive Case: left와 right로 my_list를 나누어준다.
    left = my_list[:len(my_list) // 2]
    right = my_list[len(my_list) // 2:]

    # Recursive Case: my_list를 잘게 쪼개는 과정을 재귀적으로 반복하고, merge 함수를
 사용하여 합쳐준다.
    return merge(merge_sort(left), merge_sort(right))

some_list = [11, 3, 6, 4, 12, 1, 2]
sorted_list = merge_sort(some_list)
print(sorted_list)
```

☑ 수업을 완료하셨으면 체크해주세요.

수강생 Q&A 보기 (/questions?assignment_id=432&sort_by=popular)

질문하기 (/questions/new?assignment_id=432&op1=%ED%94%84%EB%A1%9C%EA%B7%

이전 강의
합병 정렬 코드 짜기 (/assignments/201)

다음 강의
알고리즘 개요 (/assignments/399)