2.

```
arr = [5, 6, 3, 2, 1, 6, 7, 8, 22, 13, 25, 35, 12, 9999]

def merge_sort(start, end):
    if end - start == 1:
        if arr[start] > arr[end]:
            arr[start], arr[end] = arr[end], arr[start]
        return

    mid = (start + end) // 2

    merge_sort(start, mid)
    merge_sort(mid, end)

    left, right = start, mid
    merged = []
    while left < mid and right < end:
        if arr[left] > arr[right]:
            merged.append(arr[right])
            right += 1
        else:
            merged.append(arr[left])
            left += 1

    while right < end:
        merged.append(arr[right])
        right += 1

    while left < mid:
        merged.append(arr[left])
        left += 1

    arr[start:end] = merged


merge_sort(0, len(arr) - 1)
print(arr)
```

- Merge Sort를 Python을 이용해서 **재귀적**으로 구성했다.
- 최대한 index로 접근하여 메모리 사용을 줄여보았다.
- `O(nlogn)`의 시간 복잡도를 가진다.

4. 위의 소팅 알고리즘에서 수행하는 Swap 횟수는 최대 몇 번인가?

- 조사 결과, Merge Sort는 swap이 일어나지 않는다고 한다.
- https://www.quora.com/Is-there-any-swap-operation-in-merge-sort

6. 트리

```
# 재진
def recursion(node, ancestors, generations):
    if node in tree:
        for i, child in enumerate(tree[node], start=1):
            recursion(child, ancestors + [node], generations + [i])
    else:
        ancestors += [node]
        rtn = f'[{str(ancestors[0]).zfill(3)}]' if not printed else '     '

        for i, ancestor in enumerate(ancestors[1:], start=1):
            if ancestor in printed:
                rtn += '   |        '
            else:
                sibling_count = len(tree[ancestors[i - 1]])
                sibling_ranking = generations[i]
                number = str(ancestor).zfill(3)

                if sibling_count == 1:
                    rtn += f' ----- [{number}]'
                elif sibling_ranking == 1:
                    rtn += f' --+-- [{number}]'
                elif sibling_count == sibling_ranking:
                    rtn += f'   L-- [{number}]'
                else:
                    rtn += f'   +-- [{number}]'
                printed.add(ancestor)
        print(rtn)


tree = {}
edges = list(map(int, input().split()))
for i in range(0, len(edges) - 1, 2):
    tree[edges[i]] = tree.get(edges[i], []) + [edges[i + 1]]

printed = set()
recursion(edges[0], [], [1])
#node  #anc #gen
```

```
[030] --+-- [054] --+-- [001] --+-- [101]
        |           |           L-- [102]
        |           L-- [003] ----- [103]
        +-- [002]
        L-- [045] ----- [123]
```

- 4명에서 코드를 분석하며 트리를 출력