5. 재귀

2번 (Merge Sort Time Complexity Proof)

- 병합 정렬에 대해서 조별로 학습해보세요.
  - 다음 주에 Merge Sort에 대해 배울 예정이지만 미리보는 병합 정렬이라고 생각해주세요.
- Merge Sort가 진행되는 과정을 직접 그려보고 각 단계에서 어떤 일이 일어나는지 생각해보세요.
- 기초 수식에서 학습한 재귀식으로 표현하면 어떻게 되는지 같이 생각해보세요.

- 문제 2: Merge Sort, 크기 n인 배열을 입력으로 받아,
  배열을 절반으로 두개로 나눈 후,
  각 작은 배열을 재귀적으로 정렬하고,
  그 결과를 Merge한다. Mergesort의 수도 코드를 간략하게 작성해보고 시간 복잡도를 증명하시오.

```
def merge_sort(arr):
    n = len(arr)

    if n <= 1:                  # 요소가 1개 이하면 정렬할 필요x
        return

    mid = n // 2                # 중앙 인덱스
    left_group = arr[:mid]      # 중앙을 기준으로 왼쪽 리스트
    right_group = arr[mid:]     # 중앙을 기준으로 오른쪽 리스트
    merge_sort(left_group)      # 왼쪽 리스트 정렬
    merge_sort(right_group)     # 오른쪽 리스트 정렬

    left, right, now = 0, 0, 0

    # 왼쪽, 오른쪽 리스트 병합
```

```
    while left < len(left_group) and right < len(right_group):
        if left_group[left] < right_group[right]:   # 왼쪽 오른쪽 중 왼쪽 값이 작다면
            arr[now] = left_group[left]             # 왼쪽 값 먼저 넣어줌
            left += 1
            now += 1
        else:                                       # 오른쪽 값이 더 작다면
            arr[now] = right_group[right]           # 오른쪽 값 먼저 추가
            right += 1
            now += 1

    # 위에서 왼쪽, 오른쪽 한번에 하고 남아있는 자료들
    while left < len(left_group):
        arr[now] = left_group[left]
        left += 1
        now += 1

    while right < len(right_group):
        arr[now] = right_group[right]
        right += 1
        now += 1

nums = [6, 8, 4, 9, 10, 1]
merge_sort(nums)
print(nums)
```

- 증명
  - n개의 리스트를 쪼갤 때 드는 시간 복잡도

    ⇒ n개의 리스트를, 각각 1개가 될 때까지 2로 나누는 연산이 필요하므로

    ⇒ 한 번 분할하면 n/2, n/2 ⇒ 즉 2 개의 배열이 생기고, 두 번 분할하면 n/4, n/4, n/4, n/4 ⇒ 즉 4개의 배열이 생긴다. 따라서, log(n)번 분할 시 n개의 부분 배열이 생김을 알 수 있다.

    ⇒ 이를 수식으로 표현하면 다음 식의 시간복잡도이다.

    ⇒ $T(n) = T(\frac{n}{2}) + 1$

    ⇒ 기초 수식 문제에서 증명하였듯이 위의 시간복잡도는 log(n)이다.

    ⇒ 이를 한 단계씩 병합할 때마다, 리스트 전체의 숫자들을 한 번씩 읽는 과정이 필요하므로 한 단계당 n의 시간복잡도를 가진다.

    ⇒ 즉, 병합 정렬의 시간복잡도는 logn(단계의 수) * n(배열의 길이)인 O(nlogn)이 성립된다.

6번

- 문제 6: 루트 있는 트리를 입력으로 받아 아래와 같이 출력하는 알고리즘을 작성하라. 트리의 각 노드에는 1,000 미만의 자연수가 저장되어 있다. 트리의 노드 연결 관계는 다음과 같이 표현해야 한다. 아래 출력에서 루트에는 자식이 3개 있고 그 자식들 중 하나는 더 이상 자식이 없는 것임을 알 수 있을 것이다.

```
[030]--+--[054]-----[001]
       +--[002]
       L--[045]-----[123]
```

```
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 = [30, 54, 30, 2, 30, 45, 54, 1, 54, 3, 45, 123, 1, 101, 1, 102, 3, 103]
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])
```

- 예시로 주어진 코드를 보고 이해하려고 노력했다

참고자료

http://www.bowdoin.edu/~ltoma/teaching/cs231/fall16/Lectures/02-recurrences/recurrences.pdf

https://www.cs.princeton.edu/courses/archive/spr07/cos226/lectures/04MergeQuick.pdf