1. merge_sort (부분 배열의 시작주소, 길이).

* merge_sort는 항상 같은 크기의 부분 배열로 분할된다.

그래서 시작점과 길이만 알면 부분 배열에 대해서만 정렬하면 된다.

* &arr[5] == (arr+5) 이다.

- 절근.

① 탈출조건 : 원소가 2개 미만 (1 or 0)으로 쪼개지는경우.

② mid를 길이/2 (중간 인덱스값)으로 잡는다.

③ mid를 기준으로 반씩 쪼개어야 하므로 I, J라는 각 배열을 위한 변수를 만든다. ⇒ 왼쪽 부분배열은 0에서, 오른쪽은 mid에서 시작하는 mid-1, len-mid의 길이를 각각 갖는다.

④ merge_sort는 '분할'을 먼저 수행한다.

⇒ 배열의 시작 주소와 길이를 넘겨주며 2번의 재귀호출을 한다.

⑤ 여기까지 왔으면 분할은 끝났고 합칠일만 남음.

⇒ 왼쪽, 오른쪽을 비교하며 작은 순서대로 버퍼에 넣는다.

(작은쪽을 I, J중에서 ++ 해주기)

* I < mid, J < len 이라는 범위를 넘어서는 안된다.

⑥ I는 mid-1이 될때까지,

J는 len-1이 될때까지 버퍼에 넣어준다.

⑦ 각 부분 배열의 [0]이 시작 인덱스이므로 걱정 없이 [0]부터 옮겨 담아준다.

끝.

2. qSort (원 배열, 시작 인덱스, 마지막 인덱스).

* 가운데를 피벗으로 잡아야 속도 저하가 덜하다. (중앙값).

-절근.

① 탈출 조건 : left와 right가 같거나 (길이 0).
left가 이미 right 보다 커질 경우.

② 피벗을 기준으로 양쪽 부분 배열의 인덱스가 될 l, r을
l은 left-1, r은 right+1로 하나 여유 있게 잡는다.

* 증가 연산자 (++)를 전위 연산으로 쓰기위함.
⇒ 멈출 위치가 목적지가 되는 장점.

③ mid는 중앙값으로 잡는다. (중앙 인덱스가 아닌 중앙 값이 pivot!!)
↳ pivot을 기준으로 쪼개지 않는다. 오히려 l >= r을 기준으로 쪼갬.

④ ++l은 mid보다 크거나 같은 위치에서 stop.
--r은 mid보다 작거나 같은 위치에서 stop.
↳ 최대 mid까지만 진행하도록 막음.

⑤ l >= r이 되었다면 l, r 교환 종료.

⑥ swap. ⟶ 다시 ④로.

⑦ [left ~ l-1], [r+1 ~ right]로 쪼개어 호출.

* l > r이라면 상관X지만
l == r이라면 [l==r]의 요소는?
⇒ 왼쪽의 부분배열 오른쪽 끝 or 오른쪽 부분배열의 왼쪽 끝 어디에 놓아도 상관 없다. 가운데 있으니까

ex)