```
.syntax unified
.cpu cortex-m0
.thumb



.data
.align 4
Input_data: .word 2, 0, -7, -1, 3, 8, -4, 10
  .word -9, -16, 15, 13, 1, 4, -3, 14
  .word -8, -10, -15, 6, -13, -5, 9, 12
  .word -11, -14, -6, 11, 5, 7, -2, -12

Output_data: .word 0, 0, 0, 0, 0, 0, 0, 0
  .word 0, 0, 0, 0, 0, 0, 0, 0
  .word 0, 0, 0, 0, 0, 0, 0, 0
  .word 0, 0, 0, 0, 0, 0, 0, 0
// 고정값 바꾸지 말 것


.section .text
.globl bubble_sort
.thumb_func

bubble_sort://

    movs r4, #31 // 반복횟수 저장 31번만 반복해도 됨
```

```
counter_loop: // 32번 반복 외부 루프로
    movs r5, #0 // 내부 비교 루프를 위한 변수 하나 더 저장 처음부터 계속 반복함
    ldr R0, =Input_data // input 첫번째 값의 주소 r0에 저장

compare_loop: // 순차적으로 정렬하기 위한 루프
    LDR R1, [R0] // inputdata의 첫번째값 r1에 저장
    LDR R2, [R0, #4] // 두번째값 r2에 저장 메모리 1칸 이동 4byte
    CMP R1, R2 // r1,r2를 비교함
    BGT swap // r2가 작다면 swap으로 --branch swap 함수를 따로 만듦
    ADDS R0, R0, #4 // 한칸씩 이동해서 비교
    ADDS R5, R5, #1 // 비교 횟수 증가
    CMP R5, R4//r5(내부 루프의 반복 횟수가 전체 반복횟수보다 큰가? 확인)
    BLT compare_loop // R5 <= R4일 때 다시 반복

    SUBS R4, R4, #1//외부루프 횟수 감소
    CMP R4, #0//r4가 0보다 큰지 확인
    BGT counter_loop//0보다 작아진다면 out 아니라면 다시 반복

    B copy_loop//inputdata -> outputdata로 복사하기 위해 copy loop로 브랜치 이동

swap: // swap으로 branch
    MOV R3, R1//r3값에 r1저장
    MOV R1, R2//r1값에 r2저장
    MOV R2, R3//r2값에 r3저장해서 r1,r2를 swap해줌
    STR R1, [R0] // 그 후 바뀐 첫 번째 값 input_data에 저장 //input_data의 값이 계속
```

ldr 되기 때문에 input_data의 메모리에 접근해 계속 바꿔줌

STR R2, [R0, #4] // 바뀐 두 번째 값 저장

B compare_loop

// 정렬 완료 후 Output_data로 복사

copy_loop://input_data의 값을 output_data로

MOVS R6, #0 // 0부터 31까지 반복해야하므로 0으로 저장

LDR R4, =Input_data // Input_data 시작 주소

LDR R5, =Output_data // Output_data 시작 주소

copy_next://copy 시작

CMP R6, #32 // 모든 데이터 복사 완료?

BGE end_program //32보다 클때 프로그램 정지

LSLS R7, R6, #2 // R6을 왼쪽 2비트 시프트(4배)해 4바이트 단위의 메모리 오프셋 계산

LDR R0, [R4, R7] // 정렬된 데이터 가져오기

STR R0, [R5, R7] // Output_data에 저장

ADDS R6, R6, #1 // r6 인덱스 증가

B copy_next//

end_program://끝