Problem 1:
Convert the decimal integer -43 to 2-digit two's complement hexadecimal. (20 points)

ANSWER: 0xd5

\[ 43 = 2 \times 16 + 11 = 0x2b \]
\[ -43 = ff - 2b + 1 = 0xd5 \]

Problem 2:
Convert the 8-bit two's complement binary integer 0101 1101 to decimal. (20 points)

ANSWER: 93

\[ 0101 1101 = 64 + 16 + 8 + 4 + 1 = 93 \]

Problem 3:
You're given two 4-digit, 2's complement hexadecimal numbers \( X = \text{0xd141} \) and \( Y = \text{0xe503} \). Compute \( X - Y \). Remember to indicate overflow if it occurs. Show all intermediate steps clearly. (30 points)

ANSWER: 0xec3e

\[ -Y = ffff - e503 + 1 = 1afd \]
\[ X - Y = X + (-Y) = \text{0xd141} + 0x1afd = 0xec3e \]
4) Circle a language: C++ or Java

Write a function/method in that language with the following prototype/header:

```c
void reverseList(int list[], int count)
```

`list[]` is an array of integers. The number of elements in `list[]` is `>= count`. (For Java programmers, `list[]` is allocated `list.length` elements, but some of the elements may be unused; `count <= list.length`)

`reverseList()` reverses the order of the first `count` integers of the array `list[]`. Write efficient code; obviously inefficient code will be penalized. **You are not allowed to call any library function that will reverse the array for you.**

For example:

```c
int x[] = {2, 17, 10, 13, 20, -100};

reverseList(x, 4); // reverse the order of the first 4 elements in x[]
// i.e., reverse the order of 2, 17, 10, 13
```


ANSWER:

```c
void reverseList(int list[], int count) {
    // your code continues here...
    int i;
    for (i = 0; i < count / 2; i++) {
        temp = list[i];
        list[i] = list[count - i - 1];
        list[count - i - 1] = temp;
    }
}
```