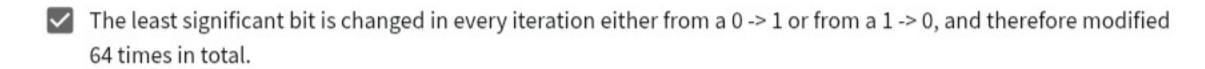
Consider the example of a binary counter shown in the lecture. Suppose the counter has 6 bits and starts with all 0s initially. Select all true facts about the problem of repeatedly incrementing the counter 64 times.

Let us represent the counter's bits as: [b5, b4, b3, b2, b1, b0] with b5 as the most significant bit and b0 as the least significant bit.

## Select all the true facts from the list below.



- **⊘** Correct
- The amortized cost of each increment operation is that of performing approximately 2 bit changes whereas the worst case for any single operation can involve as many as 6 bits changing.
- **⊘** Correct
- In the worst case, a single increment can cause all 6 bits to change in value.
- **⊘** Correct
- After the 64 increment operations are done, the counter resets back to all 0s.
- **⊘** Correct
- ▼ The total number of cumulative bit modifications for all 64 increments is given by 2 + 4+ 8 + ...+ 64 = 126 in total.
- **⊘** Correct
- The most significant bit b5 is modified exactly twice during the 64 increment operations.
- Correct
  It is modified once after the first 32 operations and second during the very last increment that resets it back to
  0.
- On the average over all 64 increments, each increment operation modifies less than 2 bits.
  - **⊘** Correct
- lacksquare Bit  $b_i$  is modified  $2^{(6-i)}$  times, for  $i=0,\ldots,5$ .
  - **⊘** Correct

2. Consider the example of a binary counter shown in the

lecture, where we will perform decrements as well as increments. Select the correct facts from the list below.

To decrement a binary counter with bits  $[b_{n-1}, \ldots, b_1, b_0]$ , we work as follows:

- 1. Scan from b0 to the left until we encounter the rightmost bit  $b_i$  that is a 1.
- 1.1 If no 1 bit is encountered, then the counter has all 0s, simply convert it to all 1s.
- 1.2 Otherwise, flip the rightmost 1 bit to 0 and make all the 0 bits to its right 1s.

Note how this is similar to the increment algorithm we looked at in the lecture and the book.

Answer questions about the amortized complexity of a binary counter that we can both increment and decrement.

- Starting from the initial counter 00..0, if we kept alternating between decrement and increment, each operation will cost n bit flips
- Correct
  Correct -- we will keep going from 00...0 to 11...1 and then from 111...1 to 000...0.
- The very same amortized analysis we used for increment can now be used to prove that the amortized complexity of increment and decrement is bounded by 2.
- The words case amortized complexity is Θ(n)
  - **⊘** Correct
- The worst case complexity of a decrement operation can be as much as n bits since decrementing 000..0 yields the value 111...1

Let D be a data structure we are designing with some operations  $o_1, \ldots, o_n$ . We will always start from an initial data structure  $D_0$ . To enable amortized analysis, we design a potential function P(D) for a given data structure instance D such that  $P(D_0) = 0$ .

## Select all the true facts from the list below.

- O The value of potential function P(D) for a data structure D obtained through a sequence of operations starting from  $D_0$  can be negative.
- The amortized cost of an operation is given by the change in potential function as a result of the operation.
- The amortized cost of an operation is given by the actual cost of the operation plus the change in potential function as a result of the operation.
- Amortized analysis using potential function guarantees that sequence of data structure operations, the sum of amortized costs of each operation is a **lower bound** to the sum of the actual cost of the operations.