Bit Operations

In the following questions, use bit manipulation operations to achieve the intended functionality and fill out the function details -

(a) Implement a function `isPalindrome` which checks if the binary representation of a given number is palindrome. The function returns true if and only if the binary representation of `num` is a palindrome. Assume `num` is 32 bits.

For example, the function should return true for `isPalindrome(0xDEADDAED)` since binary representation of 9 is `1001` which is a palindrome.

```java
/**
* Returns true if binary representation of num is a palindrome
*/
public static boolean isPalindrome(int num) {

    _________________________________________________

    _________________________________________________

    _________________________________________________

    _________________________________________________

    _________________________________________________

    _________________________________________________

    _________________________________________________
}
```

Solution:

Here is a video walkthrough for part a.

```java
/**
* Returns true if binary representation of num is a palindrome
*/
public static boolean isPalindrome(int num) {
    // stores reverse of binary representation of num
    int reverse = 0;

    // do till all bits of num are processed
    int k = num;
    while (k > 0) {
        // add rightmost bit to reverse
        reverse = (reverse << 1) | (k & 1);
        k = k >> 1;             // drop last bit
```

```
    }
    return num == reverse;
}
```

Explanation: The main idea is to reverse the bits of `num`; it is a palindrome if and only if it is equal to its reverse. To do this, we initialize `reverse` to all zeros. Inside the loop:

1. Shift `reverse` to "vacate" its last bit.
   `rrr << 1 -> rrr0`
2. Get the last bit of `k`.
   `kkkk & 0001 -> 000k`
3. `or` the numbers together to get the combined bits.
   `rrr0 | 000k -> rrrk`
4. Remove the bit of `k` we just used.

(b) Implement a function `swap` which for a given integer, swaps two bits at given positions. The function returns the resulting integer after bit swap operation.

For example, when the function is called with inputs `swap(31, 3, 7)`, it should reverse the 3rd and 7th bits from the right and return 91 since 31 (00011111) would become 91 (01011011).

```
/**
* Function to swap bits at position a and b (from right) in integer num
*/
public static int swap(int num, int a, int b) {
    _________________________________________________

    _________________________________________________

    _________________________________________________

    _________________________________________________

    _________________________________________________

    _________________________________________________

    _________________________________________________

    return num;
}
```

Solution:

Here is a video walkthrough for part b.

```
/**
* Function to swap bits at position a and b (from right) in integer num
*/
public static int swap(int num, int a, int b) {
    int p = a-1;
    int q = b-1;

    int bit_a = (num >> p) & 1;
    int bit_b = (num >> q) & 1;

    if (bit_a != bit_b) {       // if the bits are different
        num ^= (1 << p);
        num ^= (1 << q);
    }
    return num;
}
```

Explanation: To get the `k`th bit from the right in a number, we can shift the number right by `k - 1` bits, then perform an `&` with 1. For a visualization, suppose we are trying to get the third bit from the right for $b_4b_3b_2b_1$. First, we right shift by 2 to get $00b_4b_3$. $00b_4b_3$ & 0001 gives $000b_3$ as desired. This is the operation performed in line 8 and 9.
We only need to swap if the two bits are different. If the bits are different, this problem reduces to flipping the bits at position `a` and `b`. To flip a bit at position `k`, we simply `xor` it with 1 ($1 \oplus 1 = 0, 0 \oplus 1 = 1$). This corresponds to lines 12 and 13.