1 Asymptotic analysis

In the following questions, give the runtime of the following functions in $\Theta(.)$ or $O(.)$ notation as requested. Your answer should be a function of $N$ that is as simple as possible with no unnecessary leading constants or lower order terms. (MT2, Spring 2015)

(a) public static void f1(int n) {
    for (int i = 0; i < 2*n; i += 1) {
        System.out.println("hello");
    }
}

Runtime: $\Theta(n)$

(b) public static void f3(int n) {
    if (n == 0) { return; }
    f3(n/3);
    f1(n);
    f3(n/3);
    f1(n);
    f3(n/3);
}

Runtime: $\Theta(n \log n)$

(c) public static void f4(int n) {
    if (n == 0) { return; }
    f4(n-1);
    f1(17);
    f4(n-1);
}

Runtime: $\Theta(2^n)$

(d) public static void f5(int n, int m) {
    if (m <= 0) {
        return;
    } else {
        for (int i = 0; i < n; i += 1) {
            f5(n, m-1);
        }
    }
}

Runtime: $\Theta(n^m)$

(e) public static void f6(int n) {
    if (n == 0) { return; }
    f6(n / 2);
    f6(n / 2);
    f6(n / 2);
    f6(n / 2);
    g(n); // runs in $\Theta(N^2)$ time
}

Runtime: $\Theta(n^2 \log n)$
(f) public static void f7(int n, int m) {
    if (n < 10) {
        return;
    }
    for (int i = 0; i <= n % 10; i++) {
        f7(n / 10, m / 10);
        System.out.println(m);
    }
}

2 More analysis

For each problem, give the best and worst-case runtimes in \( \Theta(.) \) notation as a function of \( N \). Your answer should be simple with no unnecessary leading constants or summations.

(a) public static void removeIndex(int[] arr, int i) {
    // Assume i > 0
    int N = arr.length;
    for (int j = i; j < N; j += 1) {
        arr[j - 1] = arr[j];
    }
}

Best Case: \( \Theta(1) \)  
Worst Case: \( \Theta(N) \)

(b) public static int recurse(int N) {
    return helper(N, N / 2);
}

private static int helper(int N, int M) {
    if (N <= 1) {
        return N;
    }
    for (int i = 1; i < M; i *= 2) {
        System.out.println(i);
    }
    return helper(N - 1, M) + helper(N - 1, M);
}

Best Case: \( \Theta(2^N \log N) \)  
Worst Case: \( \Theta(2^N \log N) \)

(c) public static boolean find(int tgt, int[] arr) {
    int N = arr.length;
    return find(tgt, arr, 0, N);
}

private static boolean find(int tgt, int[] arr, int lo, int hi) {
    if (lo == hi || lo + 1 == hi) {
        return arr[lo] == tgt;
    }
    int mid = (lo + hi) / 2;
    for (int i = 0; i < mid; i += 1) {
        System.out.println(arr[i]);
    }
    return arr[mid] == tgt || find(tgt, arr, lo, mid) || find(tgt, arr, mid, hi);
}

Best Case: \( \Theta(N) \)  
Worst Case: \( \Theta(N^2) \)

CS 61B, Spring 2020, Discussion 7: Exam Prep
3 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.

For example, the function should return true for isPalindrome(9) 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) {
    // 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;
    }
    return num == reverse;
}
```

(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 6th bit from the right and return 91 since 31 (00011111) would become 91 (01011011).

```java
/**
 * 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;
}
```