1 Basic Algorithmic Analysis

For each of the following function pairs $f$ and $g$, list out the $\Theta$, $\Omega$, $O$ relationships between $f$ and $g$, if any such relationship exists. For example, $f(x) \in O(g(x))$.

1. $f(x) = x^2$, $g(x) = x^2 + x$
2. $f(x) = 500000x^3$, $g(x) = x^5$
3. $f(x) = \log(x)$, $g(x) = 5x$
4. $f(x) = e^x$, $g(x) = x^5$
5. $f(x) = \log(5^x)$, $g(x) = x$

2 Practice with Runtime

For each of the following functions, find the Big-Theta expression for the runtime of the function in terms of the input variable $n$.

You may find the following relations helpful:

\[ 1 + 2 + 3 + 4 + \cdots + N = \Theta(N^2) \]
\[ 1 + 2 + 4 + 8 + \cdots + N = \Theta(N) \]

1. For this problem, you may assume that the static method constant runs in $\Theta(1)$ time.

```java
public static void bars(int n) {
    for (int i = 0; i < n; i += 1) {
        for (int j = 0; j < i; j += 1) {
            System.out.println(i + j);
        }
    }
    for (int k = 0; k < n; k += 1) {
        constant(k);
    }
}
```
2. Determine the runtime for barsRearranged.

```java
public static void cowsGo(int n) {
    for (int i = 0; i < 100; i += 1) {
        for (int j = 0; j < i; j += 1) {
            for (int k = 0; k < j; k += 1) {
                System.out.println("moove");
            }
        }
    }
}

public static void barsRearranged(int n) {
    for (int i = 1; i <= n; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            cowsGo(j);
        }
    }
}
```

3. A Bit on Bits

Recall the following bit operations and shifts:

1. Mask (x & y): yields 1 only if both bits are 1.
   
   \[01110 \& 10110 = 00110\]

2. Set (x | y): yields 1 if at least one of the bits is 1.
   
   \[01110 \mid 10110 = 11110\]

3. Flip (x ^ y): yields 1 only if the bits are different.
   
   \[01110 ^ 10110 = 11000\]

4. Flip all (~ x): turns all 1’s to 0 and all 0’s to 1.
   
   \[\sim 01110 = 10001\]

5. Left shift (x << left_shift): shifts the bits to the left by left_shift places, filling in the right with zeros.
   
   \[10110111 << 3 = 10111000\]

6. Arithmetic right shift (x >> right_shift): shifts the bits to the right by right_shift places, filling in the left bits with the current existing leftmost bit.
   
   \[10110111 >> 3 = 11110110\]
   \[00110111 >> 3 = 00000110\]

7. Logical right shift (x >>> right_shift): shifts the bits to the right by right_shift places, filling in the left with zeros.
   
   \[10110111 >>> 3 = 00010110\]
Implement the following two methods. For both problems, \( i = 0 \) represents the least significant bit, \( i = 1 \) represents the bit to the left of that, and so on.

1. Implement \texttt{isBitOn} so that it returns a boolean indicating if the \textit{ith} bit of \texttt{num} has a value of 1. For example, \texttt{isBitOn(2, 0)} should return \texttt{false} (the 0th bit is 0), but \texttt{isBitOn(2, 1)} should return \texttt{true} (the 1st bit is 1).

\begin{verbatim}
/** Returns whether the \textit{ith} bit of \texttt{num} is a 1 or not. */
public static boolean isBitOn(int num, int i) {
    int mask = 1 _________________________________;
    return _________________________________;
}
\end{verbatim}

2. Implement \texttt{turnBitOn} so that it returns the input number but with its \textit{ith} significant bit set to a value of 1. For example, if \texttt{num} is 1 (1 in binary is 01), then calling \texttt{turnBitOn(1, 1)} should return the binary number 11 (aka 3).

\begin{verbatim}
/** Returns the input number but with its \textit{ith} bit changed to a 1. */
public static int turnBitOn(int num, int i) {
    int mask = 1 _________________________________;
    return _________________________________;
}
\end{verbatim}