Homework #1

P1: Which of the following is equivalent?

1. 3^7 and 7^3?
2. 8^2 and 4^3?
3. $16^{ln(4)}$ and $4^{ln(16)}$

P2: Prove that the sum of the first n odd integers is n^2.

P3: Prove that $\sum_{k=0}^{n} \binom{n}{k} = 2^n$ for $n \geq 0$. Hint: $\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$.

P4: By finding the limit, determine the most appropriate Landau symbol to describe the relation between $f(n)$ and $g(n)$, eg. $f(n) = O(g(n))$

1. $f(n) = n + ln(n)$ and $g(n) = 2n$
2. $f(n) = ln(n) + 5$ and $g(n) = 2lg(n)$
3. $f(n) = 2n^3 + n$ and $g(n) = 1000 \cdot ln(n) + 3n + 2$
4. $f(n) = 3n^{lg(3)}$ and $g(n) = 2n^3$

P5: The codes below is an implementation for finding Valid Triangle Number. Given an unsorted array of positive integers, find the number of triangles that can be formed with three different array elements as three sides of triangles. For a triangle to be possible from 3 values, the sum of any of the two values (or sides) must be greater than the third value (or third side).

```
// Function to count all possible triangles with arr[]
// elements
int findNumberOfTriangles(int arr[], int n)
{
    // Sort the array elements in non-decreasing order
    sort(arr, arr + n);
    // Initialize count of triangles
    int count = 0;
    // Fix the first element. We need to run till n-3
    // as the other two elements are selected from
    // arr[i+1...n-1]
    for (int i = 0; i < n - 2; ++i) {
        // Initialize index of the rightmost third
        // element
        int k = i + 2;

        // Fix the second element
        for (int j = i + 1; j < n; ++j) {
            // Find the rightmost element which is smaller
            // than the sum of two fixed elements The
            // important thing to note here is, we use the
            // previous value of k. If value of arr[i] +
            // arr[j-1] was greater than arr[k], then arr[i]
            // + arr[j] must be greater than k, because the
            // array is sorted.
            while (k < n && arr[i] + arr[j] > arr[k])
                ++k;

            // Total number of possible triangles that can
            // be formed with the two fixed elements is
            // k - j - 1. The two fixed elements are arr[i]
            // and arr[j]. All elements between arr[j+1]/ to
            // arr[k-1] can form a triangle with arr[i] and
            // arr[j]. One is subtracted from k because k is
            // incremented one extra in above while loop. k
            // will always be greater than j. If j becomes
            // equal to k, then above loop will increment k,
            // because arr[k]
            // + arr[i] is always greater than arr[k]
            if (k > j)
                count += k - j - 1;
        }
    }

    return count;
}
```

What is its time complexity and space complexity? Assume the sorting take $O(nlogn)$ runtime and no extra space.