Some practice problems to help with learning algorithm complexity and Big-O

1) Using the definition of Big O prove the following functions g(n) are O(f(n)) for the given g(n) and f(n).
   a. g(n) = 18 * n^3 + 13n, f(n)=n^3; **prove**: g(n) is O(n^3)
   b. g(n) = 34 + log2n, f(n)=log2n; **prove**: g(n) is O(log2n)
   c. g(n) = log2n + n, f(n) = n; **prove**: g(n) is O(n)
   d. g(n) = (n^2 + 1) / (n + 1), f(n) = n; **prove**: g(n) is O(n)

2) Show that 2^n is O(3^n) but 3^n is not 2^n

3) Give a big-oh upper bound on the running time of the for-loop that includes function func2(n) whose big-oh upper bound is O(f(n)).

   ```
   for ( int i = 1; i <= n - 3; i++ )
   {
       func2( n );
   }
   ```

4) What is the order of each of the following tasks in the worst case?
   a. Computing the sum of the first n even integers by using a for loop
   b. Displaying all n integers in an array
   c. Computing the sum of the first n even integers by using recursion
   d. Computing the sum of the first n even integers by using a closed formula
   e. Finding an element in an unsorted list
   f. Finding an element in a sorted list

5) The following fragment of code computes the matrix multiplication of a[n][n] and b[n][n]. Give a big-oh upper bound on the running time.

   ```
   for ( int i = 0, i < n, i++ )
   for ( int j = 0, j < n, j++ )
   {
       c[i][j] = 0.0;
       for ( int k = 0, k < n, k++ )
           c[i][j] += a[i][k] * b[k][j];
   }
   ```
6) Find a big-oh upper bound for the worst-case time required by the following algorithm. Assume that func1 is big O(f1(n)) and func2 is big O(f2(n)):

```c++
bool iskey(int s[], int n, int key)
{
    for ( int i = 0; i < n - 1; i++ )
    {
        for ( int j = i + 1; j < n; j++ )
        {
            if ( s[i] + s[j] == key )
            {
                func1(n);
            }
            else
            {
                func2(n);
            }
        }
    }
}
```

7) Let k be a positive integer. Show that \(1^k + 2^k + 3^k + \ldots + n^k\) is \(O(n^{k+1})\).
**Some Answers**

1) Using the definition of Big O prove the following functions $g(n)$ are $O(f(n))$ for the given $g(n)$ and $f(n)$.

   a. $g(n) = 18 \cdot n^3 + 13n$, $f(n) = n^3$; **prove:** $g(n)$ is $O(n^3)$

   **Answer:**

   As per the definition of BigO:
   - An Algorithm $A$ is order $f(n)$: Denoted $O(f(n))$
     - If constants $k$ and $n_0$ exist
     - Such that $A$ requires no more than $k \times f(n)$ time units to solve a problem of size $n \geq n_0$

   Let’s find a $k$ and $n_0$ so that $kn^3 > 18 \cdot n^3 + 13n$ for all $n \geq n_0$

   First, let’s divide each size by $n^3$
   
   $k > 18 + 13/n^2$

   Let’s set $k = 18+13 = 31$; and substitute in for $k$.
   
   $31 > 18 + 13/n^2$
   $13 > 13/n^2$
   $13n^2 > 13$
   $n^2 > 1$
   
   $n > 1$ so let $n_0 = 1$

1c. $g(n) = \log_2 n + n$, $f(n) = n$; **prove:** $g(n)$ is $O(n)$

   Let’s find constants $k$ and $n_0$ such that $kn > \log_2 n + n$ for all $n \geq n_0$

   $2^{kn} > 2^{(\log_2 n + n)}$
   $2^{kn} > 2^{(\log_2 n)} \times 2^n$
   
   Let $k = 2$

   $2^{2n} > 2^{(\log_2 n)} \times 2^n$
   $2^n > 2^{(\log_2 n)}$
   $2^n > n$
   
   True for $n > 1$
4) What is the order of each of the following tasks in the worst case?
   a. Computing the sum of the first n even integers by using a for loop
      Answer: \( O(n) \)
   b. Displaying all n integers in an array
      Answer: \( O(n) \)
   c. Computing the sum of the first n even integers by using recursion
      Answer: \( O(n) \)
   d. Computing the sum of the first n even integers by using a closed formula
      Answer: \( O(1) \)
   e. Finding an element in an unsorted list
      Answer: \( O(n) \)
   f. Finding an element in a sorted list
      Answer: depends on searching algorithm. Let’s say we have a variant of binary search. Then \( O(\log n) \).

The following fragment of code computes the matrix multiplication of \( a[n][n] \) and \( b[n][n] \). Give a big-oh upper bound on the running time.
```
for ( int i = 0, i < n, i++ )
   for ( int j = 0, j < n, j++ )
   {
      c[i][j] = 0.0;
      for ( int k = 0, k < n, k++ )
         c[i][j] += a[i][k] * b[k][j];
   }
```
Answer: \( O(n^3) \)

7) Let k be a positive integer. Show that \( 1^k + 2^k + 3^k + \ldots + n^k \) is \( O(n^{k+1}) \).
   \textbf{Hint:} Represent \( n^{k+1} \) as \( (n^k + n^k + n^k + \ldots + n^k) \)