Problem 1 (7 pts)

Order the following functions by growth rate. Indicate which functions grow at the same rates. 
\( N, \sqrt{N}, N^{1.5}, N^2, N \log N, N \log \log N, N \log^2 N, N \log(N^2), 2/N, 2^N, 2^{N/2}, 37, N^2 \log N, N^3 \)

Problem 2 (18 pts)

For this problem, you will need to write some code in Java. We’ve provided everything you need to get started in the Java skeleton file located at [http://www.cs.washington.edu/education/courses/cse373/13sp/homework/hw02/HW2Prob2.java](http://www.cs.washington.edu/education/courses/cse373/13sp/homework/hw02/HW2Prob2.java)

For each of the following six program fragments:

Give an analysis of the running time. Big-Oh will suffice.

Then, implement the code in Java, and give the running time (in milliseconds) for the several values of \( n \) listed in the table below. We’ve set up the skeleton files to make this easier: Look for an ”INSERT YOUR CODE HERE” comment; that is where you will add your code. The skeleton is set up to read the value of \( n \) from the command line (e.g. `java HW2Prob2 2000`).

<table>
<thead>
<tr>
<th>Big-Oh</th>
<th>( n=20 )</th>
<th>( n=200 )</th>
<th>( n=2000 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Finally, using the completed table above, compare your analysis with the actual running times and discuss.
The six fragments:

1. sum = 0;
   for (i=0; i<n; i++)
      sum++;

2. sum = 0;
   for (i=0; i<n; i++)
      for (j=0; j<n; j++)
         sum++;

3. sum = 0;
   for (i=0; i<n; i++)
      for (j=0; j<n*n; j++)
         sum++;

4. sum = 0;
   for (i=0; i<n; i++)
      for (j=0; j<i; j++)
         sum++;

5. sum = 0;
   for (i=0; i<n; i++)
      for (j=0; j<i*i; j++)
         for (k=0; k<j; k++)
            sum++;

6. sum = 0;
   for (i=1; i<n; i++)
      for (j=1; j<i*i; j++)
         if (j % i == 0)
            for (k=0; k<j; k++)
               sum++;

Problem 3 (8 pts)

Consider the following algorithm (known as Horner’s rule) to evaluate \( f(x) = \sum_{i=0}^{N} a_i x^i \):

poly = 0;
for( i = n; i >= 0; i--)
   poly = x * poly + a[i];

1. Show how the steps are performed by this algorithm for \( x = 3, f(x) = 4x^4 + 8x^3 + x + 2 \) by filling out the table. Remember that the array \( a[] \) contains the coefficients of the various powers of \( x \).
2. What is the running time of this algorithm? Give your answer in Big-Oh form and explain how you reached that conclusion.

**Problem 4 (5 pts)**

Show that the function $6n^3 + 30n + 403$ is $O(n^3)$.

You will need to use the formal definition of $O(f(n))$ to do this (see Weiss p29). In other words, find values for $c$ and $n_0$ such that the definition of Big-Oh holds true as we did with the examples in lecture.

**Problem 5 (8 pts)**

Given the following recursive search function, prove by induction that it correctly returns 1 if the value `val` is in the array `v` and 0 otherwise. (Hint: try working out all the possibilities for arrays of `size = 1` to get a sense of how your proof should proceed.)

```c
int search(v[]: integer array, size: integer, val: integer)
    if (size == 0) return 0;
    else
        if (v[size-1] == val) return 1;
        else return search(v, size-1, val);
```

You will need to provide at least these details in a complete proof:

**Basis:** The case where `size = 0`

**Inductive Hypothesis:** Assume...

**Inductive Step:**