Complexity: in principle

In all problems below, functions will be from $\mathbb{N}$ to $[0, \infty)$.

1. ***Big-O notation***: if f and g are functions, consider the statement that $f(n) = O(g(n))$.
   (a) What is the logical *definition* of this statement?
      (i) Briefly explain the purpose of each quantifier and predicate in the definition.
      (ii) How can we interpret the meaning of this concept graphically (in the sense of classical graphs of functions, not digraphs)?
   (b) What is the *point* of Big-O notation?
   (c) Semantic flaws:
      (i) In what way is the "$=$" in this expression misleading?
      (ii) What is a little shady about the n's in this expression?
   (d) How does Big-O notation work with respect to addition and subtraction?
   (e) How does Big-O notation work with respect to multiplication, and how is this significantly different than the previous case?

2. Discuss the meanings of the two related notations below; how does each relate to Big-O notation?
   (a) ***Big-*Ω** notation $f(n) = \Omega(g(n))$
   (b) ***Big-*Θ** notation $f(n) = \Theta(g(n))$

3. What is the hierarchy of sizes for basic expressions in n when n is large, and how does it guide our usage of the above asymptotics?

. . . and in practice

4. Asymptotically simplify the following expressions using Big-O notation.
   (a) $5n^3 + 3n + 1000$ (b) $6\log(n^3) + 2n + 5$ (c) $8n^{100} + 2^n + \log n$ (d) $100^n + 4n^{50} + n!$
   (e) $(100n + n^4)(n^2 + 2^n)$ (f) $(\log\log n + 10000)(\log n + n + \sqrt{n})$
   (g) $(n! + n^n + 1000^n)(10 + n^3 + 300n^2)$

5. Prove the following statements (you may use trial and error and/or a calculator to help with the $\exists$'s!):
   (a) $1000 = O(\log n)$
   (b) $100n = O(n^2)$
   (c) $3^n = \Omega(100 \cdot 2^n)$
   (d) $100n^2 = \Theta(n^2)$

[Problem set continues on back side.]

6. Suppose that you have an array of $n = 1023$ numbers, $a_0, a_1, a_2, \ldots, a_{1022}$,
   and that you want to search for a given number A in that array.
   Suppose, for the sake of this problem, that A does *not* match any element of the array.
   (a) If you perform a *linear search*, in which you compare A to each element of your array, how many comparisons will be performed during the search?
   (b) Suppose now that the elements of your array are sorted, and you do a *binary search*, comparing A to the middle number remaining at each step. How many comparisons will be performed? How does this relate to the number 1023?
      (i) What are the specific elements in the array you'll compare A to if A ends up being larger than all of them?
      (ii) What are the specific elements in the array you'll compare A to if A ends up being smaller than all of them?
   (c) Now suppose that your array has $n = 2^k - 1$ elements.
      How many comparisons will each of these algorithms use?
      What are the Big-O bounds for the complexities of each of these algorithms?

7. Suppose that you have numbers arranged on a 5×5 grid. Starting from the top-left cell, you can choose to go down or right at each step (staying on the grid) until you end at the bottom-right, and you'd like to find the path for which the sum of the numbers you pass through is as large as possible.
   (a) The brute-force search:
      (i) How many such paths are there?
         (You will take $4 + 4 = 8$ steps; you just need to *choose* which four of these are "down.")
      (ii) Along each path, how many times will you have to add two numbers?
      (iii) How many additions will you perform in total?
   (b) Dynamic programming:
      (i) Suppose that instead of the brute-force search, you work down and right through the chart, starting at the top-left and, at each stage, updating just each next cell reachable with the sum of it and the larger of the numbers above and/or left of it (sort of like Dijkstra!).
         (Choose a 5×5 grid and actually *do* this to see how it works!)
      (ii) Count the total number of additions necessary in this algorithm.
   (c) Which of the two algorithms above is more efficient?
      (What makes the more efficient algorithm possible?)
   (d) For each of these algorithms, what would the total number of additions be for a general $n \times n$ grid?
   (e) Call a solution *infeasible* if the number of additions is more than one billion.
      At what specific n does each of the above algorithms become infeasible?
      (Feel free to use a calculator or online tool to compute the values!)