Complexity Exercises: Solutions

1. a) Worst-case running time of this code fragment expressed in terms of n:

    ```python
    n = len(vals)
    sum = 0
    for i in range(n):
        sum = sum + vals[i]
    ```

    Breakdown into primitive operations

    ```plaintext
    n = len(vals)  # len(vals): 1
    # =:          1

    sum = 0        # =:          1

    for i in range(n)):  # range(): 1
        # in:         1
        # for:        2

    sum = sum + vals[i]  # sum: 1
    # vals[i]: 1
    # +=:        1
    # =: 1
    ```

    3 primitives
    plus
    8 primitives executed n times

    which is 8n + 3

b) Why can we ignore the coefficients and constants?

As n grows, the highest order term dominates.

2. Answer: O(n)
3. For each code fragment, state its big-O complexity.

   a) 

   ```python
   for x in numlist1:
       for y in numlist2:
           print(x + y)
   ```

   **Answer: O(n²)**
   Each loop has complexity O(n) and they are nested.

   b) 

   ```python
   if i%2 == 0:
       for j in range(0,len(numlist),2):
           print(numlist[j])
   else:
       for j in range(len(numlist):
           print(numlist[j])
   ```

   **Answer: O(n)**
   Both segments of code in the if and else clauses are O(n) and only one of the them will be executed (i.e., the for loops are not nested).

   c) 

   ```python
   x = 0
   n = 50
   for i in range(n):
       x = x + i
   ```

   **Answer: O(1)**
   Since n is assigned to 50, we know that the loop will be executed a constant number of times.