```
while (!a [i].equals (element))
    i++;
```

Assume that a is an array of n elements and that there is at least one index k in 0 ... n - 1 such that a [k].equals (element).

Use Big-O notation to estimate worstTime(n). Use Big-Ω and Big-Θ notation to estimate worstTime(n). In plain English, estimate worstTime(n).

3.3 Study the following method:

```
/**
 *  Sorts a specified array of int values into ascending order.
 *  The worstTime(n) is O(n * n).
 *
 *  @param x - the array to be sorted.
 *
 */
public static void selectionSort (int [ ] x)
{
  // Make x [0 ... i] sorted and <= x [i + 1] ... x [x.length -1]:
  for (int i = 0; i < x.length - 1; i++)
  {
    int pos = i;
    for (int j = i + 1; j < x.length; j++)
      if (x [j] < x [pos])
        pos = j;
    int temp = x [i];
    x [i] = x [pos];
    x [pos] = temp;
  } // for i
} // method selectionSort
```

a. For the inner **for** statement, when i $= 0$, j takes on values from 1 to n - 1, and so there are n - 1 iterations of the inner **for** statement when i $= 0$. How many iterations are there when i $= 1$? When i $= 2$?

b. Determine, as a function of n, the total number of iterations of the inner **for** statement as i takes on values from 0 to n - 2.

c. Use Big-O notation to estimate worstTime(n). In plain English, estimate worstTime(n)—the choices are constant, logarithmic in n, linear in n, linear-logarithmic in n, quadratic in n and exponential in n.

3.4 For each of the following functions f, where $n = 0, 1, 2, 3, \ldots$, estimate f using Big-O notation and plain English:

a. $f(n) = (2 + n) * (3 + \log(n))$

b. $f(n) = 11 * \log(n) + n/2 - 3452$

c. $f(n) = 1 + 2 + 3 + \cdots + n$

d. $f(n) = n * (3 + n) - 7 * n$

e. $f(n) = 7 * n + (n - 1) * \log(n - 4)$

f. $f(n) = \log(n^2) + n$

g. $f(n) = \dfrac{(n + 1) * \log(n + 1) - (n + 1) + 1}{n}$

h. $f(n) = n + n/2 + n/4 + n/8 + n/16 + \cdots$