1. (15 points) Consider the following sorted array:

   double[] data = {1, 4, 6, 7, 9, 10, 14};

   A. If using **binary search**, how many comparisons are needed (ie, how many elements in the array do we need to examine) to determine that 9 is in the array?
   3: the elements 7, 10, and 9 are examined

   B. If using **sequential search**, how many comparisons (ie, how many elements in the array do we need to examine) are needed to determine that 9 is in the array?
   5: the elements 1, 4, 6, 7, and 9

   C. If using **sequential search**, how many comparisons (ie, how many elements in the array do we need to examine) are needed to determine that 11 is not in the array?
   7: it's not until we examine the element 14 that we can determine 11 isn't in the array

2. (25 points) Fill in the blanks in the following code segment to implement sequential search of the given array. The method should return a boolean value indicating whether or not the parameter value was found in the array.

   public static boolean seqSearch(int s){
   int[] a = {9, 12, 14, 3, 25};
   for( int i = __0__; __i < a.length_________; i++){
     if (__a[i] == s_________ ){
       return ___true__________;
     }
   }
   return ___false________ ;
   }
3. (10 points) The following code segment implements selection sort as described in the lecture video and textbook. Draw the array after each iteration of the loop (ie, at the point in the code indicated by the comment). Hint: You may not need all the spaces provided for a correct answer.

```java
int[] list = {5, 3, 8, -1};

for(int i = 0; i < list.length; i++) {
    //find minimum value
    int currentMin = list[0];
    int currentMinIndex = 0;

    for (int j = i+1; j < list.length; j++) {
        if (currentMin > list[j]) {
            currentMin = list[j];
            currentMinIndex = j;
        }
    }

    //swap min element into place
    if (currentMinIndex != i) {
        list[currentMinIndex] = list[i];
        list[i] = currentMin;
    }

    /**************draw array at this point**********/
}
```

This is the same algorithm described in the video lecture. Note that algorithm doesn't stop running at next to last iteration, even though it isn't necessary. The bounds on the loop are a.length.