Worksheet 8: Searching and Asymptotic Analysis

In preparation: Read Chapter 4 to learn more about big-oh notation.

Linear search	
Binary search	
countOccurrences	
isPrime	
printPrimes	
matMult	
SelectionSort	

In this worksheet we will be concentrating on algorithms involving loops. In this case the question to ask is how many times the inner statements in a loop are being executed, and how this quantity changes if the input size is changed. As you look at each example, fill in the values in the table at left.

```
int countOccurrences (double data [ ], int n, double testValue)
{
   int count = 0; int i;
   for (i = 0; i < data.length; i++)
      { if (data[i] == testValue)
          count++;
   }
   return count;
}
```

Our first function counts the number of occurrences of a given value in an array of data. If he execution time is proportional to the size of the input array, it is O( )?

Our second function determines if an integer variable is prime. A prime, you will recall, is a value that is divisible only by itself and 1. Note that the input here is not an array, but a single integer value. What can you say about the worst-case execution time for this function in relation to the value of this variable?

```
int isPrime (int n) {
   int I;
   for (i = 2; i * i <= n; i++) {
      if (0 == n % i) return 0;
   }
   return 1; /* 1 is true */
}
```

```
void printPrimes (int n) { int
   i; for (i = 2; i < n; i++) {
      if (isPrime(i))
         printf("Value %d is prime");
   }
}
```

When one function calls another inside of the loop, the execution time of the called function must be counted in determining the time for the loop. Here there is a simple O(n) loop that calls isPrime. So the total execution time is in the worst case O( ) ?

The classic matrix multiplication routine is a good example of nested loops. Again the question to ask is how many times the statements in the innermost loop are executed as a function of the data size (in this case,

```
void matMult (int [][] a, int [][] b, int [][] c)
   { int n = n; // assume all same size
   for (int i = 0; i < n; i++) for
      (int j = 0; j < n; j++) {
         c[i][j] = 0;
         for (k = 0; k < n; k++)
            c[i][j] += a[i][k] * b[k][j];
      }
}
```

the number of rows and columns of the input arrays).

```
void selectionSort (double * storage, int n) {
    int p, i, indexLargest;
    for (p = n – 1;  p > 0; p--) {
       indexLargest = 0;
       for (i = 1; i <= p; i++) {
          if (storage[i] > storage[indexLargest])

             indexLargest = i;

       }
       if (indexlargest != position)
          swap(storage, indexLargest, position);
    }
  }
}
```

Finally, consider selectionSort. Here the inner loop is more subtle. How is it related to the outer loop? Can you identify the pattern? How many steps will it take the first time the outer loop executes? The last? What does this tell you about the number of times the innermost if statement will be executed?

Estimate big-Oh time complexity of running the following code:

for (i = 0; i < n; i++) …	
for (i = n; i > 0; i--) …	
for (i = 0; i * i < n; i++) …	
for (i = n; i > 0; i = i / 2) …	
for (i = n; i > 0; i = i >> 2) …	
for (i = 0; i < n; i++) for (j = 0; j < n; j++)	
for (i = 0; i < n; i++) for (j = 0; j < i; j++)	
for (i = 0; i < n; i++) for (j = 0; j < 13; j++)	
for (i = n; i > 0; i = i / 2) for (j = n; j > 0; j = j / 2)	
for (i = 0; i < n; i++) for (j = n; j > 0; j = j / 2)	