**CSCI 230 – Data Structures and Algorithms
Fall 2019
Lab 1: Algorithmic complexity**

**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Exercise 1: Fill in the blanks by choosing the proper word from the ones that are recommended below.**

i. **An algorithm \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ a function.**

a. Defines,
b. implements,
c. uses,
d. complements.

ii. **An algorithm must always \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

a. Start,
b. terminate

iii. **A program is a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of an algorithm.**

a. Representation,
b. synonym

iv. **An algorithm includes a series of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ steps.**

a. Complete,
b. concrete,
c. abstract,
d. special.

v. **The steps of an algorithm are \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.**

a. Ordered,
b. un-ordered.

vi. **An algorithm implements a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.**

a. Program
b. Function

c. Object

**Exercise 2 (From: Shaffer textbook)
Determine complexity for following code segments in the average case. Assume all variables are of type int.**

(a)
a = b + c;

d = a + e;

(b)
sum = 0;

for (i=0; i<3; i++)

 for (j=0; j<n; j++)

 sum++;

(c)

sum=0;

for (i=0; i<n\*n; i++)

 sum++;

(d)

for (i=0; i < n-1; i++)

 for (j=i+1; j < n; j++) {

 tmp = AA[i][j];

 AA[i][j] = AA[j][i];

 AA[j][i] = tmp;

 }

(e)

sum = 0;

for (i=1; i<=n; i++)

 for (j=1; j<=n; j\*=2)

 sum++;

(f)

sum = 0;

for (i=1; i<=n; i\*=2)

 for (j=1; j<=n; j++)

 sum++;

(g) Assume that array A contains n values, Random takes constant time,

and sort takes n log n steps.

for (i=0; i<n; i++) {

 for (j=0; j<n; j++)

 A[j] = DSutil.random(n);

 sort(A);

}

 (h) Assume array A contains a random permutation of the values from 0

to n−1.

sum = 0;

for (i=0; i<n; i++)

 for (j=0; A[j] != i; j++)

 sum++;

(i)

sum = 0;

if (EVEN(n))

 for (i=0; i<n; i++)

 sum++;

else

 sum = sum + n;

**Exercise 3: Give the best lower bound that you can for the following code**

**fragment, as a function of the initial value of n.**

while (n > 1)

 if (ODD(n))

 n = 3 \* n + 1;

 else

 n = n / 2;