Thomas Nguyen
COEN 179
Professor Walden

Assignment #1

1. Consider the following nested loops:

```
for(int i=1; i <= N; i++)
    for(int j=1; j <= i; j++)
        for(int k=1; k <= i*log(j); k++) x=i+j+k;
```

Using a timer or counting steps, calculate runtimes using
N=10, 20, 40, 100, 200, 400, 1000, 2000, 4000, 10000.

Based on the results, conjecture the complexity of the number of assignments performed. Prove your conjecture.

N = 10

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
469 steps
```

N = 20

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
5202 steps
```

N = 40

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
54090 steps
```

N = 100

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
1121240 steps
```

N = 200

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
10716155 steps
```

N = 400

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
100027099 steps
```

N = 1000

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
1862909197 steps
```

N = 2000

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
16734665574 steps
```

N = 4000

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
148588177889 steps
```

N = 10000

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
2626318282409 steps
```

Complexity

N log (N * (N+1) / 2)

Proof

```
for(int i=1; i <= N; i++)
   for(int j=1; j <= i; j++)
      for(int k=1; k <= i*log(j); k++) x=i+j+k;
```

2. Consider this program :
    a. What does the outputted value signify?

The outputted value is the square root of the input

   b. By timing or counting steps using several different inputs, conjecture the complexity of this algorithm.

```
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
Input number 1
1.000
0 steps
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
Input number 25
5.000
5 steps
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
Input number 100
10.000
6 steps
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
Input number 225
15.000
7 steps
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
Input number 400
20.000
7 steps
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
Input number 625
25.000
8 steps
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
Input number 900
30.000
8 steps
Thomass-MacBook-Pro:COEN 179 thomasnguyen$ ./a.out
Input number 1125
33.541
8 steps
```

The complexity of this algorithm is logarithmic because on each iteration the program is cutting the amount of steps it needs to talk by approximately half.

   c. Can you prove your answer?

   d. Would your answer change if the value of 3 in the two commented lines changed to 6 or 9?

Yes, the output and number of steps would change if we replaced the current value of 3 to be 6 or 9. However, the complexity of the algorithm remains LogN because we are still splitting the number of steps in half on each iteration.

3. How many additions do the recursive and iterative Fibonacci programs perform in calculating the nth Fibonacci number?

Recursive:
2^n additions

$f(0) = 0$ [0 additions]
$f(1) = 1$ [0 additions]
$f(2) = f(1) + f(0) + 1$ [1 additions]
$f(3) = f(2) + f(1) + 1$ [2 additions]
$f(4) = f(3) + f(2) + 1$ [4 additions]

Iterative:
n-1 additions

Store the previous two outputs of the Fibonacci Sequence
$[0,1] = 0+1 = 1$ [1 additions]
$[1,1] = 1+1 = 2$ [2 additions]
$[1,2] = 1+2 = 3$ [3 additions]
$[2,3] = 2+3 = 5$ [4 additions]
$[3,5] = 3+5 = 8$ [5 additions]

1.4.5 Give tilde approximations for the following quantities:

a. $N + 1$
b. $1 + 1/N$
c. $(1 + 1/N)(1 + 2/N)$
d. $2N^3 - 15 N^2 + N$
e. $\lg(2N)/\lg N$
f. $\lg(N^2 + 1) / \lg N$
g. $N^{100} / 2^N$

a) ~ N
b) ~ 1
c) ~ 1
d) ~ $2N^3$
e) ~ 1
f) ~ 1
g) ~ 1

1.4.6 Give the order of growth (as a function of N) of the running times of each of the following code fragments:

a.
```
int sum = 0;
for (int n = N; n > 0; n /= 2)
  for(int i = 0; i < n; i++)
     sum++;
```

b.
```
int sum = 0;
for (int i = 1 i < N; i *= 2)
    for (int j = 0; j < i; j++)
        sum++;
```

c.
```
int sum = 0;
for (int i = 1 i < N; i *= 2)
   for (int j = 0; j < N; j++)
       sum++;
```

a) N
   a. (N + N/2 + N/4 + N/8 …) = 2N-1
b) N
   a. (1 + 2 + 4 + 8 + 16 …) = 2N-1
c) NLogN
   a. (N + 2N + 4N + 8N + 16N …) = NlogN