Homework 1

CIS-675 DESIGN AND ANALYSIS OF ALGORITHMS

PROF. IMANI PALMER

4/17/2021

Anthony Redamonti
SYRACUSE UNIVERSITY

Question 1:

Write a small program to determine all pairs of positive integers (a,b) such that a < b < n and (a^2 + b^2 + 1)(ab) is an integer. What is the complexity of the program? Measure the runtime of the program and plot it for values of n = 10; 100; 1000 and 10000. In order to receive full points for this question, show measure of runtime for each n value, as well as, answer the question on the complexity of the program.

All positive integers $a < b < n$ start when $a = 1$ and $b = 2$ since 1 is the smallest positive integer and b must be greater than a.

The python program below prints all pairs of positive integers (a,b) such that $a < b < n$ and $(a^2 + b^2 + 1)(ab)$ is an integer.

```
import time
import random

def algorithm(n):
   count = 0
   for a in range(1, n):
      for b in range(a+1, n):
         print(a,b)
   return

times = []
N = 10
while (N <= 10000):
   StartTime = time.time()
   algorithm(N)
   EndTime = time.time()
   TotalTime = (EndTime - StartTime)
   times.append(TotalTime)
   N*=10

N = 10
for TotalTime in times:
   print('N = ', N)
   N *= 10
   print("Total Time: {0}".format("{:.{}f}".format(TotalTime, 8)))
```

The portion of the output related to the runtime is shown below.

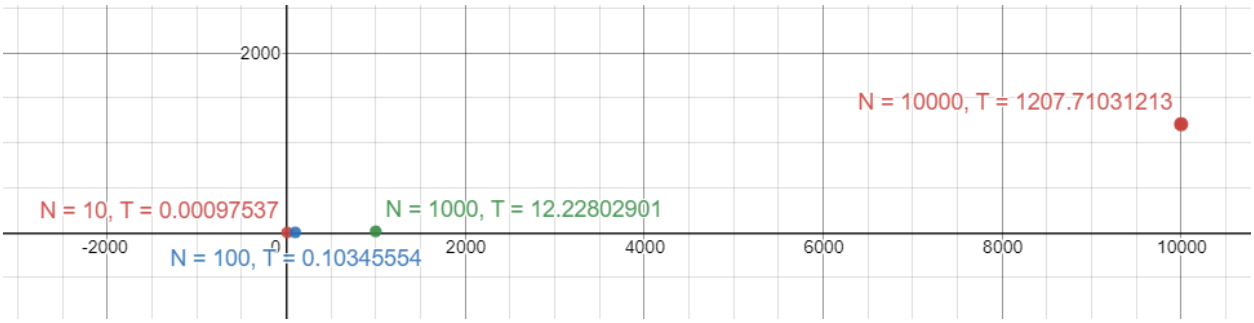
```
N =  10
Total Time: 0.00097537
N =  100
Total Time: 0.10345554
N =  1000
Total Time: 12.22802901
N =  10000
Total Time: 1207.71031213

Process finished with exit code 0
```

In a table, these points are as follows:

N	Runtime (seconds)
10	0.00097537
100	0.10345554
1,000	12.22802901
10,000	1207.71031213

The following is a plot of the input (N) vs. its runtime (T). (N, T)



"answer the question on the complexity of the program"

$$for(int\ a = 1;\ i < N;\ i++)\{$$
$$\quad for(int\ j = a + 1;\ j < N;\ j++)\{\ldots\}$$
$$\}$$

Iteration	i	Inner-Loop Iterations
0	1	N-2
1	2	N-3
2	3	N-4
3	4	N-5
4	5	N-6

5	6	N-7
k	k+1	$N-(k+2)$
$N-2$	N-1	$N-((N-2)+2)=0$

The series in the table can be written as:

$$\sum_{k=0}^{N-2}(N-(k+2))=(N-k-2)$$

We know that:

$$\sum_{k=0}^{N}(N-k)=\frac{N(N+1)}{2}$$

Knowing that:

$$\sum_{k=0}^{N}(N-k-2)=\frac{N(N+1)}{2}-N-N-1=\frac{N(N+1)}{2}-2N-1=\frac{N^2+N}{2}-2N-1$$

We only care about the high-order term. Therefore, the complexity of the program is $O(N^2)$.

Question 2:

Consider two arrays (the first array and the second array) of the same size, each consisting of n random positive integers (each integer between 1 and n). Write a program to remove duplicates between two arrays. If a positive integer exists in both arrays, replace it with zero in the second array. And then return total number of zeros in the second array. In order to receive full points for this question, show measure of runtime for array sizes of 10, 100, 1000, and 10000, as well as, answer the question on the complexity of the program.

The python program below creates two arrays “array1” and “array2” each of size N containing random positive integers between 1 and N. The algorithm uses two for-loops to iterate through each array. Each element of array1 is compared to each element of array2. If a duplicate element is found, then that element is set to 0 in array2. Then the total number of duplicates is returned from the algorithm.

```
import time
import random

def algorithm(array1, array2, n):
   number_of_zeroes = 0
   for i in range(0, n):
      for j in range(0, n):
         if(array1[i] == array2[j]):
            array2[j] = 0
            number_of_zeroes += 1
   return number_of_zeroes

N = 10
while (N <= 10000):
   print('N = ', N)
   array1 = [N]
   array2 = [N]
   for i in range(N):
      array1.append(random.randint(1, N))
      array2.append(random.randint(1, N))

   StartTime = time.time()
   zeroes = algorithm(array1, array2, N)
   EndTime = time.time()
   TotalTime = (EndTime - StartTime)
   print('number of zeroes ', zeroes)
   print("Total Time: {0}".format("{:.{}f}".format(TotalTime, 8)))
   N*=10
```

The output of the program is below.

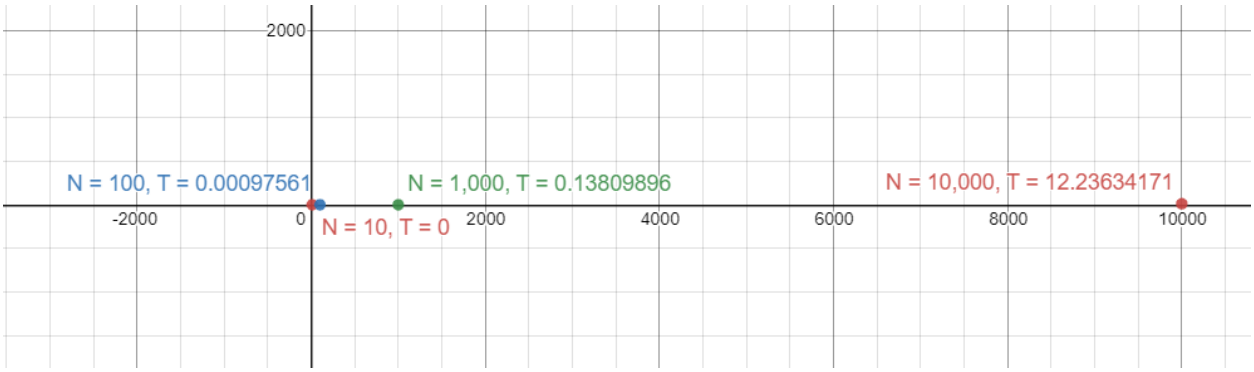
```
homework1question2
C:\Python38\python38.exe "C:/Users/aredamonti/projects/Des
N =  10
number of zeroes  9
Total Time: 0.00000000
N =  100
number of zeroes  64
Total Time: 0.00097561
N =  1000
number of zeroes  629
Total Time: 0.13809896
N =  10000
number of zeroes  6338
Total Time: 12.23634171

Process finished with exit code 0
```

In a table, these points are as follows:

N	Runtime (seconds)	Number of Duplicates
10	0.00000000	9
100	0.00097561	64
1,000	0.13809896	629
10,000	12.23634171	6338

The following is a plot of the input (N) vs. its runtime (T). (N, T)



"answer the question on the complexity of the program"

$$for(int\ i = 0;\ i < N;\ i++)\{$$
$$\quad for(int\ j = 0;\ j < N;\ j++)\{\ldots\}$$
$$\}$$

$$Outer\ Loop\ Runtime: \frac{N-0}{1} = O(N)$$

$$Inner\ Loop\ Runtime: \frac{N-0}{1} = O(N)$$

Since the two loops have no dependencies on each other's variables, the total runtime is the product of each loop's runtime.

$$Total\ Runtime = O(N^2)$$

Question 3:

1. $for(int\ i = 1; i < N; i \mathrel{*}= 3)\{\dots\}$

   $Loop\ will\ run\ until\ 3^k \geq N, where\ k\ is\ the\ number\ of\ iterations\ in\ the\ loop.$

   $$Applying \log_3 to\ both\ sides:\ k = \log_3 N$$

   $$Runtime = O(\log(N))$$

2. $for(int\ i = N; i > 1; i \mathrel{/}= 2)$

   $$Loop\ will\ run\ until \frac{N}{2^k} \leq 1, where\ k\ is\ the\ number\ of\ iterations\ in\ the\ loop.$$

   $$Simplifies\ to: N = 2^k$$

   $$Applying \log_2 to\ both\ sides:\ k = \log_2 N$$

   $$Runtime = O(\log(N))$$

3. $for(int\ i = 1; i < N; i \mathrel{+}= 2)\{$
   $for(int\ j = 1; j < N; j \mathrel{*}= 2)\{\dots\}$
   $\}$

   $$Outer\ Loop\ Runtime: \frac{N-1}{2} = O(N)$$

   $$Inner\ Loop\ will\ run\ until\ 2^k \geq N$$

   $$\log_2(2^k) = \log_2(N)$$

   $$k = \log_2(N)$$

   $$Inner\ Loop\ Runtime: \log_2(N) = O(\log(N))$$

$Since\ the\ two\ loops\ have\ no\ dependencies\ on\ each\ other's\ variables, the\ total\ runtime\ is\ the\ product\ of\ each\ loop's\ runtime.$

$$Total\ Runtime = O(Nlog(N))$$

4. $for(int\ i = 1; i < N; i \mathrel{*}= 2)\{$
   $for(int\ j = i; j < N; j++)\{\dots\}$
   $\}$

Iteration	i	Inner-Loop Iterations
0	1	N-1
1	2	N-2
2	4	N-4
3	8	N-8
4	16	N-16
5	32	N-32
k	2^k	$N - 2^k$

$Log_2(N)$	$2^{Log_2 N} = N$	N-N = 0

The series in the table can be written as:

$$\sum_{k=0}^{Log_2 N} (N - 2^k)$$

We know that:

$$\sum_{k=0}^{Log_2 N} 2^k = \frac{2^{\log_2(N)+1} - 1}{2 - 1} = \frac{2N - 1}{1} = 2N - 1$$

Substituting $2N - 1$ in for 2^k yields (N-(2N-1)) = 1-N

$$Total\ Runtime = O(N)$$

Question 4:

For the following algorithm, state the best and worst-case scenarios. State the runtime of each in terms of Big O of N.

```
void mySort(int arr[], int n){
        int index = 0;
        while(index < n){
                if(index==0){
                        index++;
                }
                if(arr[index] >= arr[index-1]){
                        index++;
                }
                else{
                        swap(arr[index], arr[index-1]);
                        index--;
                }
        }
        return;
}
```

For the following algorithm, state the best and worst-case scenarios. State the runtime of each in terms of Big O of N.

The best-case scenario is when the array "arr" is already sorted. In this case, the algorithm will search through each element in the array, comparing it to its preceding element. The runtime in this case is $O(N)$.

The worst-case scenario is when the array "arr" is sorted in reverse order (greatest to smallest). In this case, the algorithm must perform the swap function on the element and its predecessor and then travel in the negative direction to compare the preceding element with its predecessor. The first index in this case would be visited N times. The runtime in this case is $O(N^2)$.