CS 210-002 Assignment 1

Name: Dustin Guest

ID: 200241546

Instructor: Dr. Maher Elshakankiri

Date: Jan. 18 2020

1.5 Write a recursive function that returns the number of 1 in the binary representation of N. Use the fact that this is equal to the number of 1 in the representation of N/2, plus 1, if N is odd.

**int binCount(unsigned int N)**

**{**

**If (N == 0)**

**return 0;**

**return N%2 + binCount(N/2);**

**}**

1.10 What is 2100 (mod 5)?

**1**

2.1 Order the following functions by growth rate: N, √ N, N 1.5 , N 2 , N logN, N log logN, N log2N, N log(N2 ), 2/N, 2N, 2N/2 , 37, N2 logN, N3 . Indicate which functions grow at the same rate.

**37, 2/N, √N, N, N log logN, N logN, N logN2, N log2N, N1.5, N2, N2logN, N3, 2N = 2N/2**

2.3 Which function grows faster: N logN or N 1+ E / √ logN, E > 0?

**N logN for large numbers**

2.7 For each of the following six program fragments:

a. Give an analysis of the running time (Big-Oh will do).

b. Implement the code in the language of your choice, and give the running time for several values of N.

c. Compare your analysis with the actual running times.

(1) sum = 0;

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

++sum;

1. **O(n)**

(2) sum = 0;

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

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

++sum;

1. **O(n2)**

(3) sum = 0;

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

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

++sum;

1. **O(n3)**

(4) sum = 0;

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

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

++sum;

1. **O(n2)**

(5) sum = 0;

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

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

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

++sum;

**a. O(n5)**

(6) sum = 0;

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

for( j = 1; j < i \* i; ++j )

if( j % i == 0 )

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

++sum;

1. **O(n5)**

2.10 Determine, for the typical algorithms that you use to perform calculations by hand, the running time to do the following:

a. Add two N-digit integers.

**Algorithm**

1. **start**
2. **i = 0,**
3. **add all the 10i digits together**
4. **i++**
5. **if i < N: go to 3**
6. **end**

**Runtime: O(N)**

b. Multiply two N-digit integers.

**Algorithm**

1. **start**
2. **i = 0**
3. **j = i**
4. **multiply the 10j digit from the first integer by the 10i digit from the second integer**
5. **j++**
6. **while j < N: go to 4**
7. **i++**
8. **while i < N: go to 3**
9. **end**

**Runtime: O(N2)**

1. Divide two N-digit integers.

**Algorithm**

1. **start**
2. **I = N - 1**
3. **Look at the 10i digit of the dividend,**
4. **Find the multiple of the divisor that is less than the 10i digit of the dividend**
   1. **This might require going through all the single digit multiples of the divisor (10N steps)**
5. **The factor of that multiple is the 10i digit of the quotent**
6. **Subtract the multiple x 10i from the dividend**
   1. **Worst case this means subtracting an N-digit number from dividend (N steps)**
7. **Decrement i**
8. **If i <= 0: go to 3**
9. **end**

**Runtime: O(n2)**

2.20

a. Write a program to determine if a positive integer, N, is prime.

b. In terms of N, what is the worst-case running time of your program? (You should be able to do this in O( √ N).)

**3 + √N = O(N)**

1. Let B equal the number of bits in the binary representation of N. What is the value of B?

**B = logN +1**

1. In terms of B, what is the worst-case running time of your program?

**3 + 2B-1/2 = O(2B)**

1. Compare the running times to determine if a 20-bit number and a 40-bit number are prime.

**B = 20, O(2B) = 1024**

**B = 40, O(2B) = 104827**

1. Is it more reasonable to give the running time in terms of N or B? Why?

**It I more reasonable to express the running times in N. N is more understandable to us and gives us more information. When B increses by 1 this can translate to huge increses in N. For example when B increase from 20 to 21, N could be increasing by Millions. We also cant expect the client to know binary, if the client wanted to know the affects of doubling the size of N, B would not be vary helpful.**

2.27 The input is an N by N matrix of numbers that is already in memory. Each individual row is increasing from left to right. Each individual column is increasing from top to bottom. Give an O(N) worst-case algorithm that decides if a number X is in the matrix.

**Algorithm**

1. **start**
2. **i = 0**
3. **if matrix[i][i] == X: return true, go to 8**
4. **if matrix[i][i] > X:**
   1. **j = i**
   2. **decrement j**
   3. **if matrix[i][j] == X: return true, go to 8**
   4. **if matrix[j][i] == X: return true, go to 8**
   5. **while j > 0: go to 4.b.**
   6. **return false, go to 8**
5. **i++**
6. **while i < N: go to 3**
7. **return false**
8. **end**

**Runtime: O(N)**