CS202 Fundamental Structures of Computer Science II Quiz 1 Name : _____ ID : _____ Section : 1	1-a		20
	1-b		20
	1-c		20
	2		40
	Total		100

Important: Every incorrect answer will end up with -10 points. So, don't answer unless you are sure.

1. (60 pts) What is the running time of each of the code fragments? Give your answers in *Big-O* notation.

   (a)
   ```
   int k=0;
   for (int i=0; i < 100*n; i+=100)
      k++;
   ```
   Answer: ***O(N)***

   (b)
   ```
   int k=0;
   for (int i = n; i > 0; i/=2)
      for (int j = 2*n; j > 0; j/=2)
         k++;
   ```
   Answer: ***O(log^{2}N)***

   (c) Given an array **"A"** of size N, containing positive integers less than **10**. In order to get ith element of the array, one should use **`A[i]`**. Find the runtime of the following code fragment, that uses the array explained above.

   ```
   for (int i = 1; i <= N; ++i)
      for (int j = i, k = i; j <= N; j++)
         for(int w = 0; w < A[i] + A[j]; w++) {
            k++;
            if (k > N)
               break;
         }
   ```
   Answer: ***O(N^2)***

2. (40 pts) Consider the following recursive function. (12 pts) First write a recurrence equation that corresponds to the growth rate of this recursive function. (23 pts) Then solve this recurrence equation using the *repeated substitution method* to find the growth rate in terms of *Big-O* notation. (5 pts) What does this recursive function calculate? Explain in few words.

   ```
   int f(int n) {
         if (n == 0)
            return 0;
         return n % 10 + f(n/10);
   }
   ```

   Answer: ***(1) T (n) = T(n/10) + c = T(n/10) + O(1)***
   (2) T(N) = O(logN)
   (3) This function returns the sum of digits of given N. ex: N = 123, answer = 1 + 2 + 3 = 6.

Here are some challenging bonus problems for your interest. If 100 points are fine for you, don't worry about these bonus problems.
Please show your work to receive credits from bonus problems.

3. (BONUS 8 pts) What is the running time of each of the code fragments? Give your answers in *Big-O* notation.

(a) CANCELLED

(b) **Worst case: $O(\log_2 N)$. Best case: $O(\log_{10} N)$. Log base doesn't change complexity => *Answer: O(logN)***

```
int j=0;
   for (int i = n; i > 0; ) {
         if (j % 2 == 0)
            i = i/2;
         else
            i = i/10;
         j++;
}
```

(c) Given an array **"A"** of size N, containing positive integers less than **100.** In order to get i^{th} element of the array, one should use **`A[i]`**. Find the runtime of the following code segment, that uses the array explained above.

```
for (int i = 1; i <= N; i++)
   for (int j = A[i]; j > 0; j/=2)
      for (int k = 1; k < log(N); k *= 2)
         ;
```

Answer: ***O(N*log(logN))***

4. (BONUS 8 pts) Consider the following recursive function. (2 pts) First write a recurrence equation that corresponds to the growth rate of this recursive function. (4 pts) Then solve this recurrence equation using the *repeated substitution method* to find the growth rate in terms of *Big-O* notation. (2 pts) What does this recursive function calculate? Explain in few words.

```
int f(int a, int n) {
      if (n == 0)
         return 1;
      if (n % 2 == 1)
         return a * f(a*a, n/2);
      return f(a * a, n/2);
}
```

Answer: ***(1) T(a, N) = T (N) = T(N /2) + c = T(N/2) + O(1)***
(2) T(N) = O(logN)
(3) This function calculates a^N in logN time. It's called logarithmic exponentiation.