Practice Exam
Algorithms and Programming for High Schoolers
(AddisCoder)

Question 1: Imagine evaluating the following expressions in order in the Python interpeter. For each expression written in red, write down what the expression would evaluate to in the space below. If the expression would cause an error in Python, then write Error.

```
>>> x = 5
>>> y = 7
>>> z = 2
>>> x * y + z


>>> (x * y) + z


>>> x * (y + z)


>>> x % y


>>> y % x


>>> [] * z
```

```
>>> [] * '2'

>>> [1] * z + x

>>> str(2)

>>> [[]][0]

>>> [[]][1]

>>> len([[[]]])

>>> [[]][]

>>> x = [2, 3]
>>> y = []
>>> for i in xrange(x[0]**x[1]):
>>>     y += [i*2]
>>> y

>>> def isPrime(x):
>>>     if x < 2:  return False
>>>     for i in xrange(2, x):
```

```
>>>       if i*i >= x:  break
>>>       if x % i == 0:  return False
>>>    return True
>>> isPrime(1)

>>> isPrime(2)

>>> isPrime(9)

>>> isPrime(12)
```

Question 2: Consider the `lists` `[]`, `[[]]`, `[[[]]]`, ... The *depth* of such a `list` is the number of nested layers of brackets. So, `depth([])` is 0, `depth([[]])` is 1, `depth([[[]]])` is 2, etc. Write a function `depth`(L) which takes such a `list` and computes its depth. What's the running time of your function in terms of n, the number of brackets in the list?

Question 3: Consider the following code for testing whether a number is prime or not.

```
def isPrime(n):
    if n < 2: return False
    for i in xrange(2, n):
        if n % i == 0: return False
        return True
```

What is the running time of this code as written? Is it correct? If it's not correct, suggest a minor change to the code which would make it correct.

Question 4: Write a function `cubeRoot`(n) which takes a positive integer n and outputs the largest integer x such that $x^3 \leq n$.

(a) Give a solution with running time $O(n^{1/3})$.

(b) Give a solution with running time $O(\log_2 n)$.

The running times above are assuming you can do arithmetic on integers up to n in $O(1)$ time.

Question 5: Suppose we have a list of numbers L[0],L[1],...,L[n-1]. An *inversion* in the list is a pair i, j such that $i < j$ but L[i] > L[j]. In other words, an inversion is a pair of indices where the larger number comes before the smaller number.

Describe an algorithm for counting the number of inversions in a list, then implement your algorithm in Python as `countInversions`(L). Faster running times get more points. Hint: The problem is solvable in $\Theta(n \log_2 n)$ time, though an easier solution takes $\Theta(n^2)$ time. You get more points for giving a slow, correct solution than a fast, incorrect solution.

Question 6: We've discussed making change using the least number of coins possible. What if we want to count how many different ways there are of making change? For example, if the coins we have available are [1,5,10,25] cents and we want to make change for 12 cents, there are 4 ways: (1) give all 1-cent pieces, (2) give a 10-cent piece and two 1-cent pieces, (3) give two 5-cent pieces and two 1-cent pieces, and (4) give one 5-cent piece and seven 1-cent pieces. So, the answer in this case is 4.

Write a function `change`(L, n) which outputs the number of ways to make change for n cents when the coin denominations available are those in L. For example, `change`([1,5,10,25], 12) should return 4. What is the running time of your solution? Faster running times get more points.

Question 7: Given a directed graph where each edge has a length, describe an algorithm that takes as input two vertices u, v and an integer $k \geq 0$ and outputs the length of the shortest path from u to v which takes *exactly* k steps. The path is allowed to visit vertices multiple times (for example, the path $1 \to 3 \to 2 \to 3 \to 7$ is a valid path from 1 to 7 of length 4, even though it visits vertex 3 twice). What is the running time of your algorithm? You do not have to write the code for it.

Question 8: In class I described Karatsuba's algorithm for multiplying two n-digit numbers, which recursively multiplied three pairs of $n/2$-digit numbers then combined the results in $O(n)$ time to get an overall running time of $\Theta(n^{\log_2 3})$.

Suppose that there existed an algorithm for multiplying two n-digit numbers which recursively multiplied *two* pairs of $n/2$-digit numbers then combined the results in $O(n)$ time. What would the running time then be?