Assignment 3 CS395-S13 Due: Mar 4 at 12:30 pm PST

Name: ______________________________ Points: 115

Remember no late papers and so always turn something in for partial credit. You can turn in paper copy before class on the due date/time or as a pdf through the website any time up to the due date/time. **Don't forget to put your name on your paper** even when you turn it in electronically.

The algorithms below are described using Python as pseudocode. Some of imports may have been left out for succinctness. Some of the code is not implemented optimally but implemented for clarity.

When talking about the "order" in the questions below I am referring to a big-O upper bound. One approach to help answer questions 1 through 3 is to assume the worst case which is every number is prime. That will certainly put an upper bound on the function execution and simplify the job of answer questions 1 through 3.

1. (30 pts) Consider this algorithm for printing all the primes less than or equal to `max`:

```
# PRIMES (version 1)

def primes(max) :
    for n in range(2, max+1) :
        if isprime(n) :
           print(n)

def isprime(n) :
    if n==2 : return True
    for k in range(2, n-1) :
        if n%k == 0 : return False
    return True
```

   a) Assuming that we are concerned with execution time what is the key operator or key block of code you want to count to measure the order of execution?

   b) How do you measure the size of input?

   c) What is the order of execution? Explain how you got your answer.

2. (15 pts) Consider this version of the code:

```
# PRIMES (version 2)

def primes(max) :
    for n in range(2, max+1) :
        if isprime(n) :
            print(n)

def isprime(n) :
    if n==2 or n==3 : return True
    if n%2==0 : return False
    for k in range(3, n-1, 2) :
        if n%k == 0 : return False
    return True
```

Now what is the order of execution using the same input and key operator from the previous problem? Explain how you got your answer and account for the step size of 2 in the for loop on k.

3. (20 pts) Now consider this version of the code:

```
# PRIMES (version 3)

def primes(max) :
    for n in range(2, max+1) :
        if isprime(n) :
            print(n)

def isprime(n) :
    if n==2 or n==3 : return True
    if n%2==0 : return False
    for k in range(3, int(sqrt(n))+1) :
        if n%k == 0 : return False
    return True
```

a) Now what is the order of execution using the same input and key operator from the previous problem? Explain how you got your answer. Be sure to account for affect of the presence of the sqrt in the algorithm on your answer.

b) How does this execution time compare with the previous problem?

4. (40 pts) Now consider this version of the code:

```
# PRIMES (version 4)

def primes(max) :
    # PART 1: create an array all set to True
    n = []
    for k in range(0, max+1) :
        n.append(True)

    # PART 2: cross off composite numbers
    p=2;
    while p <= max :
        for j in range(2*p, max+1, p) :
            n[j] = False
        p+=1

    # PART 3: print what is left
    for k in range(2, max+1) :
        if n[k] : print(k)
```

a) Assume that we are concerned with execution time. This algorithm can be divided into three parts. Each has a key operation whose execution time is not dependent on the size of the input. For each of the three parts list the key operation.

b) What is the order of execution for each part of the code? Explain how you got your answer.

c) Because each key operation is independent of the input size, we can assume they remain in some constant ratio of size to each other and so we can combine the orders of execution for each of the three parts by simply adding them. What is the overall order of execution as implemented? Show how you got your answer.

d) What is the order of execution given in terms of a simple function of the input size without using addition. That is, give the simplified form of the formula for the order of execution.

5. (20 pts) If the resource is now space and not time. Compare versions 1 and 4. Specifically, what is the order of space utilization for both algorithms assuming each variable and/or array element takes 1 unit of memory. Show how you got your answer.

6. (30 pts) Consider this algorithm

```
def primes(max) :
    # PART 1: create an array all set to True
    n = []
    for k in range(0, max+1) :
        n.append(True)

    # PART 2: cross off composite numbers
    p=2;
    while p <= max :
        # cross off composites
        for j in range(2*p, max+1, p) :
            n[j] = False

        # find next prime
        p+=1
        while p<=max and n[p]==False :
            p+=1

    # PART 3: print what is left
    for k in range(2, max+1) :
        if n[k] : print(k)
```

What is the order of execution of this algorithm assuming that the "find next prime" section is of inconsequential time. Your answer must account for the fact that the next prime step is sampling primes whose density "thins out" as *max* gets larger. For simplicity assume that Gauss' prime counting[1] function is not an approximation but is exact:

$$\pi(n) = \frac{n}{\ln(n)}$$

Feel free to use the back of this page if you are turning in a paper copy.

[1]The function that counts the number of primes less than or equal to n is classically denoted: $\pi(n)$. This approximation was first noted by Gauss about 1792.