# CS 721 Assignment-1

1. Analyze the complexity of the following program. Please provide the O complexity of the code and show your reasoning.

```python
import math
def print_primes(N):
    for i in range(2, N):
        flag = 0
        for k in range(2, int(math.sqrt(i))+1):
            if(i % k == 0):
                flag = 1
                break
        if flag == 0:
            print(i)

    print('done')
```

   The first for loop iterates in range of 2 to n. So the time complexity will be n-1.
   For flag = 0 it's constant as 1
   Inner for loop iterates in range of 2 and root n plus 1. So complexity for it becomes n^0.5
   For if conditions complexity becomes constant as 1.

   Therefore, the final time complexity will be (n-1)*1*(n^0.5)*1*1 and will have an higher order of n^1.5
   i.e O(n^1.5)

2. Function Evaluations

   (i). Log^(4) 4

   We can write log^(4) 4 = log(log(log(log4)))

   We can write 4 = 2^2 and log 4 base 2 is 2

   = log(log(log(2)))

   = log(log(1))

   = log(0)

   = undefined

   (ii). Log^4 4

   We can write log^4 4 as (log 4)^4

   We know log4 base 2 is 2.

   ⇨ (log 4)^4 = 2^4 =16

   (iii). 4! √4 = (1*2*3*4)(2) = 48.

3. Prove if the following is True or False
   1.

a. $2^{n+1} = O(2^n)$

**Claim:** $f(n) = 2^{n+1} \in O(2^n)$

**Proof by definition:** To prove this claim by definition, we need to find some positive constants c and n0 such that $f(n) <= c2^n$ for all $n > n0$ . (Note: you just need to find one concrete example of c and n0 satisfying the condition.)

$2^{n+1} \leq C * 2^n, \forall n \geq 1$ and $C >= 2$.

Therefore, if we let C = 2 and n0 = 1, we have $f(n) \leq c\, 2^n$ , $\forall n \geq n0$ .

Hence according to the definition of big-Oh, $f(n) = O(2^n)$ is true.

Alternate Method: $f(n) = 2^{n+1}$ and $g(n) = 2^n$

$\lim_{n\to\infty} g(n)/f(n) \Rightarrow \lim_{n\to\infty} 2^n/2^{n+1} = ½ > 0$

Therefore $2^{n+1} = O(2^n)$ is True

b. $2^{2n} = O(2^n)$

To prove $f(n) = O(g(n))$ we have $\lim_{n\to\infty} g(n)/f(n) > 0$

Here $g(n) = 2^n$ and $f(n) = 2^{2n} = (2^2)^n = 4^n$

$\lim_{n\to\infty} g(n)/f(n) \Rightarrow \lim_{n\to\infty} 2^n/4^n = \lim_{n\to\infty} (2/4)^n = 0$

Therefore $2^{2n} = O(2^n)$ is false.

2. Show that the solution of $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ is $O(n \log n)$.

Given 1st time $T(n) = 2T(n/2 + 17) + n$

Substitute n= n/2 in RHS 2nd time

We get $T(n) = 2(T(n/2^2 + 17) + n/2) + n \Rightarrow 2T(n/2^2 + 17) + n + n \Rightarrow 2T(n/2^2 + 17) + 2n$

Again Substitute n= n/2 in RHS 3rd time

We get $T(n)\ 2(T(n/2^3 + 17) + n/2) + 2n \Rightarrow 2T(n/2^3 + 17) + n + 2n \Rightarrow 2T(n/2^3 + 17) + 3n$

Again Substitute n= n/2 in RHS 4th time

We get $2(T(n/2^4 + 17) + n/2) + 3n \Rightarrow 2T(n/2^4 + 17) + n + 3n \Rightarrow 2T(n/2^2 + 17) + 4n$

After doing for k times

We have $T(n) = 2(T(n/2^k + 17) + n/2) + (k-1)n \Rightarrow 2T(n/2^k + 17) + n + (k-1)n \Rightarrow 2T(n/2^k + 17) + kn$

Suppose $n/2^k$ tends to 1.

⇨ $2^k = n$

- ⇨ k = logn

substitute k value

we get T(n) = 2T(1 + 17) + nlogn => T(n) = 2T(18) + nlogn

As T(18) is constant we can say time complexity as O(nlogn).

3. Show that the solution of $T(n) = T(|n/2|) + 1$ $is$ $O(\log n)$.

Given $1^{st}$ time $T(n) = T(n/2) + 1$

Substitute n= n/2 in RHS 2nd time

We get T(n) = T(n/2^2) + 1 +1 => T(n/2^2) + 2

Again Substitute n= n/2 in RHS $3^{rd}$ time

We get T(n) = T(n/2^3) + 1 + 2 => T(n/2^3) + 3

Again Substitute n= n/2 in RHS $4^{th}$ time

We get T(n) = T(n/2^4) + 1 + 3 => T(n/2^4) + 4

After doing for k times

We have T(n) = T(n/2^k) + 1 + k – 1 => T(n/2^k) + k

Suppose n/2^k tends to 1.

- ⇨ 2^k = n
- ⇨ k = logn

substitute k value

we get T(n) = T(1) + logn

As T(1) is constant we can say time complexity as O(logn).