Name: Tazmeen Afroz Roll No :- 22P-9252

SECTION - BAI-5A

ASSIGNMENT - 1

Q1)

1.1

a. $f(n) = 3n^3 + 5n^2 + 7$

Sol:- Asymptotic Complexity : $O(n^3)$

In this case, the highest polynomial power is $3n^3$. As n^3 grows larger, the terms with lower powers ($5n^2$ and 7) become insignificant compared to n^3. Therefore the asymptotic complexity is $O(n^3)$

b. $(g(n) = 2^{\sqrt{n}}$

Sol:

Asymptotic Complexity : $O(2^{\sqrt{n}})$

- Exponential functions grow faster than polynomial functions
- The base of the exponent does not affect the asymptotic complexity.

(c) $h(n) = n \log^2(n)$

Asymptotic Complexity : $O(n \log^2(n))$

While $\log^2(n)$ grows slower than n, multiplying it by n results in a function that still grows faster than $\log(n)$ but slower than n^2.

(d) $k(n) = n!$

Asymptotic Complexity, $O(n^n)$

$n! = n \times (n-1) \times (n-2) \dots 2 \times 1$. Each term in the factorial grows linearly with n and there are n terms. Therefore n^n is grows faster than any other polynomial

1.2)

$f(n)$ is $O(g(n))$

Let

$f(n) = 5\log n + 100$ and $g(n) = \log n$

$f(n)$'s $O(\log n)$

$f(n) \leq c_1 g(n)$

$5\log n + 100 \leq c_1 \log n$

As n grows large, the term $3\log n$ dominates the constant term 100

There exists a constant $c > 0$ (e.g $c = 301$) and an $n_0 \geq 0$ (e.g $n_0 = 1$) such that $3\log n + 100 \leq 301 * \log n$ for all $n \geq 1$

$f(n)$ is Not $O(g(n))$

Let

$f(n) = 3n^2$ $g(n) = n$

$3n^2 \leq c_1 n$

As n grows large, $3n^2$ grows much faster than n.

There doesnot exist any constant $c > 0$ such that $3n^2 \leq cn$ for all sufficiently large n

This violates the definition of O-notation proving that $f(n)$ is not $O(n)$

1.3

$(p(n) = 5n^2 + 3n + 1)$

Soln

function is polynomial.

→ The highest power of n is 2, which is finite.

→ consists of terms raised to integer power

$q(n) = 4^n$

Sol..

function is exponential.

→ consist of an exponential term

→ base (4) is constant, and the exponent (n) varies.

$r(n) = \log(n) \cdot n$ Neither

function is neither purely polynomial nor purely exponential.

→ log (n) is a logarithmic function, not a polynomial or exponential function

$s(n) = \sqrt{n} \cdot 2^n$

function is exponential

→ n^k is polynomial and 2^n is exponential

→ exponential term will dominate the growth of f(n) as n increases.

→ so overall function will be exponential

Scanned with CamScanner

Q2)

2.1. Determine the time complexity.

Soln

```
def loop_count(n):
    count = 0                      -> (1)
    for i in range(n):             -> (n+1)
        for j in range(1,n):       -> (n) (n+1) -> n^2+n
            count += 1             -> n x n  -> n^2
    return count
```

Outer loop -> iterates n times -> n+1 -> one where fails

Inner loop iterates i -> n-1 -> n -> one where fails

Total iterations

$1+2+\dots n = n(n+1)/2$

Time Complexity is $O(n^2)$

2.2

```
def nested_loops(n):
    total = 0                          -> (1)
    for i in range(1, n):              -> (n+1)
        for j in range(i,n):           -> n (n+1) -> n^3+n
            for k in range(n):         -> n x n x (n+1)
                total += 1             -> n x n x n -> n^3
    return total
```

Outer loop :

(n+1) times

Inner loop.. (n+1)

Inner most (n+1)

Total iteration

$\sum_{i=1}^{n} \sum_{k=0}^{n-1} (n-k) =$

Time Complexity is $O(n^3)$

Scanned with CamScanner

2.3

```
def nested_count(n):
    total = 0                        -> (1)
    for i in range(n):               -> (n+1)
        for j in range(n):           -> (n)(n+1)
            for k in range(i+1):     -> (n+1) Σ(i+1)
                total += 1           -> (n+1) n(n+1)/2
    return total
```

n^3

Outer loop (n+1) times

Middle loop n+1 times

Inner loop
$\sum_{k=1}^{i} (i+1)$ times

Total $\sum_{i=1}^{n} \sum_{k=1}^{i} (i-k+1)$

Time Complexity = $O(n^3)$

2.4

```
def multi_loops(n):
    total = 0
    for i in range(1, n):          -> (n+1)
        for j in range(2i, n):     -> n x n/2
            ~~for k~~ : total += 1
    return total
```

$n^2/2$

Outer loop takes n+1 times
Inner loops takes $n/2$ times
but still we ignore constants
So time complexity will be $O(n^2)$.

Q.5

```
def multi_loops(n):
    total = 0                          → (1)
    for i in range (1,n):              → (n+1)
        for j in range (1,i):          → n x Σ_{j=1}^{n} j
            for k in range (1,j):  n x n x Σ_{k=1}^{n} k
                total += 1
    return total.
```

$$\sum_{i=1}^{n-1} \sum_{j=1}^{i-1} \sum_{k=1}^{j-1} 1$$

$$\sum_{i=1}^{n-1} \frac{i(i-1)}{2}$$

$$\sum_{i=1}^{n-1} i^2$$

$$= \frac{n(n-1)(2n-1)}{6}$$

$$= n^3$$

Time Complexity of this algorithm will be $O(n^3)$.

3.1)

Prove or disprove that the function $T(n) = 5 \cdot 2^{n/2} + 3n^2$ is $O(2^n)$

To check if $T(n)$ can upper-bounded by 2^n.

$$T(n) \le c_1 \cdot 2^n$$

$$5 \cdot 2^{n/2} + 3n^2 \le c_1 \cdot 2^n$$

$$\frac{5 \cdot 2^{n/2}}{2^n} + \frac{3n^2}{2^n} \le c_1$$

$$5 \cdot 2^{n/2 - n} + 3\frac{n^2}{2^n} \le c_1$$

$$5 \cdot 2^{-n/2} + 3\frac{n^2}{2^n} \le c_1$$

$$5 \cdot 2^{-\frac{(1)}{2}} + 3\frac{(1)^2}{2^1} \le c_1$$

$$5.03 \le c_1$$

$$5 \le c_1 \quad \text{positive value}$$

Thus we prove that $T(n) = 5 \cdot 2^{n/2} + 3n^2$ is indeed $O(2^n)$.

if equation is like this as there is confusion.

$$T(n) = \frac{5}{2} \cdot 2^n + 3n^2 \quad \text{is } O(2^n)$$

$$\frac{5}{2} 2^n + 3n^2 \leq c_1 2^n$$

$$\frac{5}{2} \frac{\cancel{2^n}}{\cancel{2^n}} + \frac{3n^2}{2^n} \leq c_1$$

$$\frac{5}{2} + 3 \frac{n^2}{2^n} \leq c_1$$

Put $n_0 = 1$

$$\frac{5}{2} + 3 \frac{(1)^2}{2^1} \leq c_1$$

$$\frac{5}{2} + \frac{3}{2} \leq c_1$$

$$\frac{\cancel{8}\,4}{\cancel{2}} \leq c_1$$

$$4 \leq c_1$$

Satisfied

So time complexity will be $O(n^2)$.