Name: Kashif Khan
Roll NO: 22P-9005
Section: BAI-5A

Assignment # 01

Q 1):-
(1.1):- a) $f(n) = 3n^3 + 5n^2 + 7$.

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

b) $(g(n)) = 2^{\sqrt{n}}$.
Solution:- 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) $n(n) = n \log^2(n)$.
Solution:- Asymptotic complexity: $O(n \log^2(n))$.
While $\log^2(n)$ grows slower than ~~log~~ n, multiplying it by by n results in a function that still grows faster than $\log(n)$ but slower than n^2.

d) $K(n) = n!$

Solution:- Asymptotic complexity : $O(n^n)$

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

(1.2) :-

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

Solution:- Let, $f(n) = 5\log n + 100$ and $g(n) = \log n$

$$f(n) = O(\log n)$$

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

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

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 $O(g(n))$

Solution:- 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 does not exist any constant $c > 0$ such that

$3n^2 \le cn$ for all sufficiently large n. This violates the definition of O-notation proving that $f(n)$ is not $O(n)$.

(1·3) :-

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

Solution:- Function is polynomial

* The highest power of n is 2, which is finite.
* Consists of terms raised to integer power.

$q(n) = 4^n$.

Solution :- Function is exponential.

* Consists of an exponential term
* Base (4) is constant and the exponent (n) varies.

$r(n) = \log(n) \cdot n$.

Solution :- Function is neither purely polynomial nor purely exponential.

* $\log(n)$ is a logarithmic function, not a polynomial or exponential function.

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

Solution :- 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.

Q 2) :-

(2.1) :- Determine the time complexity.

Solution :-

```
Def loop.count(n):
    count = 0                      → (1)
    for i in range(n):             → (n+1)
        for j in range(i,n):       → (n)(n+1) → n² n
            count += 1             → n × n → n²
    return count
```

Outer loop iterates n times → $n+1$

Inner loop iterates $i \to n-1 \to n \to$ one where fail

Total iterations

$$1+2+\dots+n = \frac{n(n+1)}{2}$$

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) → n³ + n
            for k in range(n):         → n × n × (n+1)
                total += 1             → n × n × n → n³
    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 $O(n^3)$.

(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=0}
                total += 1         → (n+1) n(n+1)/2
    return total                   → n^3
```

Outer loop: $(n+1)$ times

Middle loop: $n+1$ times

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

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

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(2*i, n):      → n × n/2
    return total                     → n²/2
```

Outer loop takes n+1 times

Inner loops takes n/2 times but still we ignore constants so time complexity will be $O(n^2)$.

(2·5) :-

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

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

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

$\Rightarrow \frac{n(n-1)(2n-1)}{6} = n^3$

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

(3.1) :- Prove or disprove that the function

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

Solution :- Check if $T(n)$ be upper bounded by 2^n

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

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

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

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

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

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

$$5.03 \leq c_1$$

$$5 \leq c_1 \quad \because \text{positive value}$$

Thus we prove that

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