Assignment 1

Question 1

1.A

```
function CheckUnique(A, n):
        for i = 1; i < n:
                for j = i + 1; j< n:
                        if a1 == aj :
                                return False
        return True
```

```
i = 1                           #assign
while(i < n):                   #compare
        j = i + 1                   #arithmetic, assign
        while(j < n)                #compare
                if(a1 == a2):           #compare
                        return False
                j++                     #arithmetic, assign
        i++                         #arithmetic, assign
return True
```

1.A.I

Primitive operations: Assign, Compare, Arithmetic
Data structures: Arrays

1.A.II

Worst-case running-time:
$T(n) = (\sum_{i=1}^{n} n)$
$= ((n-1) * (\frac{n}{2}))$
$= O(n^2)$

1.B

```
def matrix_squared(A: list[list[int]], n: int) -> bool:
        for i in range(1, n): # Compare
                for j in range(1, n): # Compare
                        for k in range(1, n): # Compare
```

```
                                C[i, j] = C[i, j] + (A[i, k] * A[k, j]) #
Access, Arithmatic, Assign
                        # Arithmatic
                # Arithmatic
        # Arithmatic
        return True
```

1.B.I

Primitive operations: Assign, Sum, Product, Access
Data structures: Arrays

1.B.II

`C[i, j] = C[i, j] + (A[i, k] * A[k, j])`
Constant time operations and their counts:

Operation	Occurrence
Assign	1
Sum	1
Access	3
Product	1

as these primitive operations are happening in the `k` loop, the running time for the `k` loop would be $T(n_k) = 8n_k \therefore T(n) = 2n * 2n * 8n = 32n^3$

1.B.III

From 1.B.II we got the running time:
$T(n) = 32n^3 + 1$

Running time using Θ-notation:
$c_1n^3 \leq 32n^3 \leq c_2n^3$
pick $c_1 = 1, c_2 = 40, n_0 = 1$
$1 \leq 32 \leq 40$
$\therefore \Theta(n^3)$ is the **Tight bound** of the running time

1.C

```
def factorial(n):
        i = 1
        j = 1
        while(j < n):
                j = j + 1
```

```
            i = i * j
    return i
```

1.C.I

Loop invariant: at the start of iteration j, it must be that $i = j!$

Initialization:
first iteration: $i = 1, j = 1$
$1 = 1!$
$1 = 1$
$\therefore$ the loop invariant is true at the start of the first iteration

Maintenance:
Assume: $i_j = j!$

$i_{j+1} = (j+1)!$
$i_j * \cancel{(j+1)} = j! * \cancel{(j+1)}$
$\therefore i = (j+1)!$

Termination:
The loop terminates when $j = n$. Due to the loop invariant at this point $i = n!$ which is the value being returned by the algorithm. $\therefore$ the algorithm correctly outputs the factorial of an input n.

1.C.II

Worst-case running time:
$T(n) = 2 + 5n = O(n)?$
Pick: $c = 7, n_0 = 1$
$2 + 5(1) \leq 7(1)$
$7 \leq 7$
$T(n) = O(n)$
$\therefore$ the worst-case running-time of this algorithm is $O(n)$

Question 2

2.A

$3n^2 + n + 10 = \Theta(n^2)$ - **TRUE**
Pick: $c_1 = 2, c_2 = 15, n_0 = 1$
$2(1)^2 \leq 3(1)^2 + (1) + 10 \leq 15(1)^2$
$2 \leq 14 \leq 15$
$\therefore 3n^2 + n + 10 = \Theta(n^2)$

2.B

$\sqrt{n} = \Omega(n^2)$ - **FALSE**

By contradiction.

Assume that c, n_0 exist:

$\sqrt{n} \geq cn^2$

$n \geq cn^4$

$\frac{1}{n^3} = c$

$lim_{n\to\infty} \frac{1}{n^3} = 0 \therefore \nexists c > 0$ such that $\sqrt{n} \geq cn^2$ which is a **contradiction**

2.C

$\frac{\lg n}{10} = \Omega(1)$ - **TRUE**

Pick: $c = 1, n_0 = 2^{10}$

$\frac{10}{10} \geq 1$

$1 \geq 1$

$\therefore \frac{\lg n}{10} = \Omega(1)$

2.D

$\lg n^8 = O(\lg n)$ - **TRUE**

Pick: $c = 9, n_0 = 10$

$8 \lg n \leq c \lg n$

$8 \leq 9$

$\therefore \lg n^8 = O(\lg n)$

2.E

$3^n = O(2^n)$ - **FALSE**

By contradiction.

Assume that c, n_0 exist:

$\frac{3^n}{2^n} \leq c\frac{\cancel{2^n}}{\cancel{2^n}}$

$\lim_{n\to\infty} \frac{3}{2}^n = \infty \therefore \nexists c > 0$ such that $3^n \leq c2^n$ which is a **contradiction**

2.F

$100n^3 = o(n^3 \lg n)$ - **TRUE**

$100 \cancel{n^3} < c \cancel{n^3} \lg n$

$100 < c \lg n$

$2^{100/c} < n$

Pick: $n_0 = 2^{101/c}$

$100 < c \lg 2^{101/c}$

$100 < 101$
$\therefore 100n^3 = o(n^3 \lg n)$

2.G

$5n^{10} = \omega(n^{10})$ - **TRUE**
$5 \cancel{n^{10}} > c \cancel{n^{10}}$
$5 > c$
Pick: $n_0 = 1$
$5(1) > c(1)$
$\therefore 5n^{10} = \omega(n^{10})$

2.I

$\log^2(n) = O(\log(n^2))$ - **TRUE**
$\frac{\cancel{\log^2 n}}{\cancel{\log(n)}} \leq \frac{2c \cancel{\log(n)}}{\cancel{\log(n)}}$
$\log(n) \leq 2c$
Pick: $c = 1, n = 10$
$1 \leq 2(1)$
$\therefore \log^2(n) = O(\log(n^2))$

2.H

$3\log_7(n) = \Theta(\log_2(n))$
Pick: $c_1 = 1, c_2 = 2. n_0 = 7$
$c_1 \lg(n) \leq 3\log_7(n) \leq c_2 \lg(n)$
$(1)\lg(7) \leq 3 \cancel{\log_7(7)} (1) \leq (2)\lg(7)$
$\lg(7) \approx 2.81$
$lg(7) \leq 3 \leq 2\lg(7)$
$\therefore 3\log_7(n) = \Theta(\log_2(n))$

2.J

$\frac{1}{n^2} = O(1)$
Pick: $c = 1, n_0 = 1$
$\frac{1}{1^2} \leq 1$
$1 \leq 1$
$\therefore \frac{1}{n^2} = O(1)$

Question 3

3.A

$f_1(n) = O(g_1(n))$, $f_2(n) = O(g_2(n))$
$f_1(n) \leq c_1 g_1(n)$, $f_2(n) \leq c_2 g_2(n)$
Pick $c = c_1 + c_2, n_0 = max(n_{01}, n_{02})$
$f_1(n) + f_2(n) \leq c_1 g_1(n) + c_2 g_2(n)$
$f_1(n) + f_2(n) \leq (c_1 + c_2)(g_1(n) + g_2(n))$
$f_1(n) + f_2(n) \leq c(g_1(n) + g_2(n))$
$\therefore$ by definition $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$

3.B

Prove that if$f(n) = O(n) + O(n^2) + O(n^3)$ then $f(n) = O(n3)$.
$f(n) = O(n) + O(n^2) + O(n^3)$
$f(n) \leq c_1 n + c_2 n^2 + c_3 n^3$
$f(n) \leq c_1 + c_2 + c_3(n + n^2 + n^3)$
Pick: $c = c_1 + c_2 + c_3, n_0 = max(n_{01}, n_{02}, n_{03})$
$f(n) \leq cn^3$
$\therefore f(n) = O(n^3)$ if $f(n) = O(n) + O(n^2) + O(n^3)$

Question 4

$T(n) = T(n-1) + 10$

4.A

Assume that the base case is $T(1) = \Theta(1)$. Show that the solution to this recurrence is $O(n)$ using:

4.A.I

Substitution method:
guess: $T(n) = O(n)$
Assume $T(k) \leq ck$ for $k < n$
$T(n) = T(n-1) + 10$
$\leq c(n-1) + 10$
$= cn - c + 10$
$-c + 10 \leq 0$
$c \geq 10$
$\therefore$ by induction $T(n) = O(n)$ when $c \geq 10$

4.A.II

Recursion tree method

```
Level 1  : T(1)   = θ(1)
Level 2  : T(2)   = T(1) + 10
```

```
Level 3  : T(3)   = T(2) + 10
...
Level n-1: T(n-1) = T(n-2) + 10
Level n  : T(n)   = T(n-1) + 10
```

There are n levels in the recursion tree and each level there is a computation time of 10. Thus the total cost at all levels of the tree is $10n$.

$\therefore$ The Running time of this recurrence using the recursion tree method is $O(n)$

4.B

YES
The recurrence $T(n) = t(n-1) + 10$ with the base case $T(1) = \Theta(1)$
preforms 10 units of constant work every function call and as there are n number of calls the total amount of work done for this algorithm is $10n$. as Ω notation is used for the **lower bound** of the running time function and the running time for this recurrence is **at least** $10n$ we can say that $T(n) = \Omega(n)$

4.C

The master method can't be used to solve this recurrence as the master method is used to solve recurrences in the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$. As the value of n is being reduced by a constant amount (1) and not a factor (b) the master method can't be used to solve this recursion.

Question 5

5.A

Level 1: n
Level 2: $2\frac{n}{4} + \frac{n}{8} = \frac{5n}{8}$
Level 3: $\frac{16n}{64} + \frac{4n}{64} + \frac{4n}{64} + \frac{n}{64} = \frac{25n}{64} = n(\frac{5}{8})^2$
...

$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$ for $|x| < 1$
$\therefore$ the sum of the cost at each level is $\frac{1}{1-\frac{5}{8}} = \frac{8}{3}n$
The upper-bound for this recurrence would be $O(n)$

5.B

The lower-bound for $T(n)$ can be found by looking at the cost of the first level of the recursion tree (n). Since every node in the tree represents some amount of work done, the total cost of the tree must be at least n $\therefore$ the lower-bound for $T(n) = \Omega(n)$

Question 6

6.A

$T(n) = 64T\left(\frac{n}{8}\right) + 3n$

$a = 64, b = 8, f(n) = 3n$
$n^{\log_b a} = n^{\log_8 64} = n^2$
$f(n) = 3n = \Omega(n)$
Case 1: $f(n) = O(n^{2-\epsilon})$ where $\epsilon = 1$
$\therefore T(n) = \Theta(n^2)$

6.B

$T(n) = 8T\left(\frac{n}{2}\right) + n^3$

$a = 8, b = 2, f(n) = n^3$
$n^{\log_b a} = n^{\log_2 8} = n^3$
$f(n) = \Omega(n^3)$
Case 2: $f(n) = \Theta(n^3 \lg^0 n)$ where $k = 0$
$\therefore T(n) = \Theta(n^3 \lg n)$

6.C

$T(n) = T(2n) + n^2$

$a = 1, b = 1, f(n) = n^2$
The master theorem can not be used for this recurrence as the running time function is not in the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ where the function is doubling $(2n)$ instead of decreasing by a factor of b and the value of $b \leq 1$

6.D

$T(n) = T\left(\frac{3n}{10}\right) + n$

$a = 1, b = \frac{10}{3}, f(n) = n$
$n^{\log_b a} = n^{\log_{\frac{10}{3}} 1} = 1$
$f(n) = \Omega(n)$
Case 3: $f(n) = \Omega(n^{0+1})$ where $\epsilon = 1$ and $\frac{3}{10}n \leq cn$ where $c = 2$
$\therefore T(n) = \Theta(n)$

6.E

$T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n}$

$a = 2, b = 2, f(n) = n^{\frac{1}{2}}$
$n^{\log_b a} = n^{\log_2 2} = n$
$f(n) = \Omega(n^{\frac{1}{2}})$
Case 1: $f(n) = O\left(n^{1-\frac{1}{2}}\right)$ where $\epsilon = \frac{1}{2}$
$\therefore T(n) = \Theta(n)$

6.F

$T(n) = T\left(\frac{n}{7}\right) + \lg^3 n$

$a = 1, b = 7, f(n) = \lg^3 n$
$n^{\log_b a} = n^{\log_7 1} = 1$
$f(n) = \Omega(\lg^3 n)$
Case 2: $f(n) = \Theta(n^0(\lg^k n))$ where $k = 3$
$\therefore T(n) = \Theta(lg^4 n)$