a) $\alpha^2 = 3\alpha - 2$

$\alpha^2 - 3\alpha + 2 = 0$

$(\alpha \quad -2) \quad \alpha_1 = 2$

$(\alpha \quad -1) \quad \alpha_2 = 1$

$T(1)=1 \quad T(2)=2$

$T(n) = C_1 . 2^n + C_2 . 1^n$

$T(1) = C_1 . 2^1 + C_2 . 1^1 \Rightarrow \cancel{-}2C_1 \cancel{+} C_2 = \cancel{-}1$

$T(2) = C_1 . 2^2 + C_2 . 1^2 \quad +4C_1 + C_2 = 2$

$2C_1 = 1$

$C_1 = \frac{1}{2} \quad C_2 = 0$

$\Rightarrow T(n) = \frac{1}{2} 2^n$

$T(n) = \theta(2^n)$

$T(2) = 2$
$T(4) = 3$
$T(8) = 4$

b) $T(n) = T(n/2) + 1 \quad T(n) = \log n$

$a = 1 \quad a = b^d$
$b = 2 \quad 1 = 2^0$
$d = 0 \quad$ so $n^0 . \log n \Rightarrow T(n) \in \theta(\log n)$

$T(n) = T(n/2) + 1$
$T(n/2) = T(n/4) + 1$
$T(n/4) = T(n/8) + 1$
$\vdots$
$T(2) = T(1) + 1$

$T(n) = T(n/2^3) + 1 + 1 + 1$
$T(n/2) = T(n/8) + 1 + 1$
$T(n) = T(n/2^k) + k$

$2^k = n$
$k = \log_2 n$

$T(n) = T(1) + \log_2 n$
$T(n) = \log_2 n + 1$
$T(n) = \theta(\log n)$

$T(2) = 2 \quad 2^k \to k+1$
$T(4) = 3$
$T(8) = 4$
$T(16) = 5$

Guess $T(n) = \log n + 1$

$\log n + 1 \ ? \ \log \frac{n}{2} + 1 + 1$

$\log n + 1 \ ? \ \log n + \log 2^{-1} + 1 + 1$

$\log n + 1 = \log n + 1$

So $T(n) = \log n$

d) $T(n) = 4T(n/2) + n^2$

$a = 4 \quad 4 = 2^2$
$b = 2 \quad a = b^d$
$d = 2 \quad$ So $T(n) \in \theta(n^2 \log n)$

e) $T(n) = 2T(n/2) + O(n)$

$a = 2 \quad a \quad b^d$
$b = 2 \quad 2 = 2^1$
$d = 1$

So $T(n) \in \theta(n \log n)$

g) $T(n) = T(n/2) + n$
$T(n/2) = T(n/4) + n/2$
$T(n/4) = T(n/8) + n/4$
$\vdots$
$T(2) = T(1) + 2$

$T(n) = T(n/8) + (n/4) + (n/2) + n$
$T(n/2) = T(n/8) + (n/4) + (n/2)$

$T(n) = T(n/2^3) + (n/2^2) + (n/2^1) + (n/2^0)$

$T(n) = T(n/2^k) + n \sum_{i=0}^{k-1} \frac{1}{2^i}$

$2^k = n$

$T(n) = T(1) + 2^k \left(1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^{k-1}}\right) \quad 1$

$2^k + 2^{k-1} + 2^{k-2} + \dots + 2^1 + 2^0 - 2^0$

$T(n) = T(1) + 2^k - 2$
$T(n) = 2n - 1 - 1 + 1$
$T(n) = 2n - 1 \quad T(n) \in \theta(n)$

h) $T(n) = 2T(\sqrt{n}) + 1 \quad T(1) = 1 \quad T(4) = 3$

$T(4) = 2T(2) + 1 \quad T(2) = 1 \quad 2^1 \quad 2^1 - 1$
$T(16) = 2T(4) + 1 \quad T(4) = 3 \quad 2^2 \quad 2^2 - 1$
$T(256) = 2T(16) + 1 \quad T(16) = 7 \quad 2^4 \quad 2^3 - 1$
$T(256) = 15 \quad 2^8 \quad 2^4 - 1$

$n = 2^k \quad T(2^k) = 2k - 1$
$k = \log_2 n$
$T(n) = 2 \log_2 n - 1$
$T(n) \in O(\log n)$

c) $T(n) = 4T(n-1) - 4T(n-2) + 3n$

$a_n^{(general)} = a_n^{(homogeneous)} + a_n^{(other)}$

$T(n)_g = 4T(n-1) - 4T(n-2)$

$\alpha^2 = 4\alpha - 4 \Rightarrow \alpha^2 - 4\alpha + 4$
$\alpha_1 = 2, \ \alpha_2 = 2$

$T(n)_g = C_1 . 2^n + C_2 . 2^n . n$

$T(n)_ö = Ax + B$

$Ax + B = 4(Ax - A + B) - 4(Ax - 2A + B) + 3x$

$Ax + B = \cancel{4Ax} - 4A + \cancel{4B} - \cancel{4Ax} + 8A - \cancel{4B} + 3x$

$Ax + B = 3x + 4A$

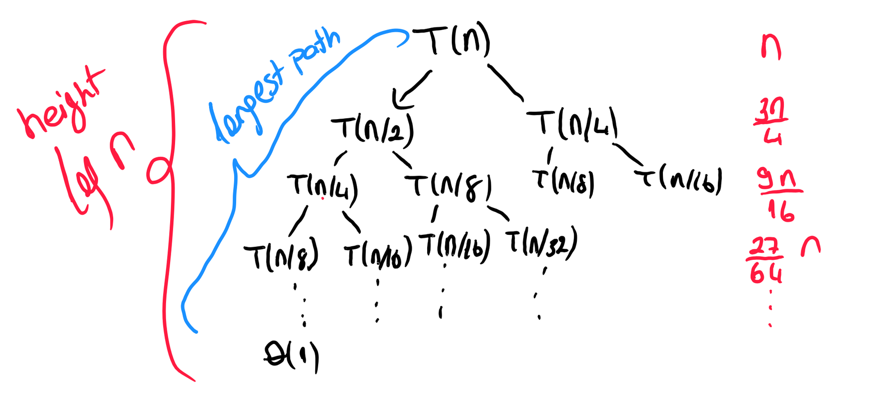
$A = 3$
$B = 12$

$T(n)_ö = 3x + 12$

$T(n) = C_1 . 2^n + C_2 . 2^n . n + 3n + 12$

$T(n) \in \theta(2^n . n)$

f) $T(n) = T(n/2) + T(n/4) + n$



n
$\frac{3n}{4}$
$\frac{9n}{16}$
$\frac{27}{64} n$

Sum of all steps

$A = \left(\frac{3}{4}\right)^0 . n + \left(\frac{3}{4}\right)^1 n + \left(\frac{3}{4}\right)^2 . n + \dots + \left(\frac{3}{4}\right)^{\log n}$

log n terms.

and each term is smaller than n

$A < n . \log n$

So $T(n) \in O(n \log n)$

Q-3)

a) $T(n) = 5T(n/2) + n^3$
b) $T(n) = 2T(n-2) + n$
c) $T(n) = 3T(n/2) + O(n^2)$

a) $T(n) = 5T(n/2) + n^3$

Master Teorem

$a = 5 \quad a < b^d$
$b = 2$
$d = 3 \quad 5 < 2^3$

So $T(n) \in O(n^3)$

b) $T(n) = 2T(n-2) + n$

$a = 2 \quad a > 1$
$b = 2$
$d = 1 \quad f(n) \ a^{n/b}$

So $T(n) \in O(n . 2^{n/2})$

c) $T(n) = 3T(n/2) + O(n^2)$

$a = 3 \quad 3 < 2^2$
$b = 2$
$d = 2 \quad$ So $T(n) \in O(n^2)$

I would prefer C, because obviously n^3 grows faster than n^2.

$\lim_{n \to \infty} \frac{n^3}{n^2} = \lim_{n \to \infty} n = \infty \quad n^2 \in O(n^3)$

So I will compare option b and c.

$\lim_{n \to \infty} \frac{n^2}{n . 2^{n/2}} = \lim_{n \to \infty} \frac{n}{2^{n/2}} = \frac{\infty}{\infty}$ L hospital $\lim_{n \to \infty} \frac{1}{2^{n/2} . \frac{1}{2} . \ln 2} = 0$

So $n . 2^{n/2}$ grows faster. $n^2 \in O(n 2^{n/2})$

So least complex one is n^2 which is option C

Q-5)

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

$T(n) = 2T(n/2) + n$
$T(n/2) = 2 . T(n/4) + n/2$
$T(n/4) = 2 . T(n/8) + n/4$
$\vdots$
$T(2) = 2 . T(1) + 2$

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

$T(n) = 2^k T(n/2^k) + \sum_{i=0}^{k-2} \frac{n}{2^i}$

$n = 2^k$

$T(n) = 2^k . T(1) + k . 2^k$

$2^k = n$
$k = \log_2 n$

$T(n) = n . \log_2 n$

Q-4)

4. The maximum cardinality matching problem in bipartite graphs involves finding the largest possible set of pairwise non-adjacent edges in a given bipartite graph. A bipartite graph is a graph in which the set of vertices can be divided into two disjoint sets, A and B, such that all edges connect a vertex from set A to a vertex in set B (and vice versa).

Provide a polynomial-time algorithm to compute a maximum cardinality matching in bipartite graphs and analyze the worst-case, best-case and average-case time complexity of the algorithm.

We initialize Maximal matching M as empty

$M = \emptyset$

While there exists an Augmenting path P

Remove matching edges of p from M and add not matching edges p to M

So this increases size of M by 1 as p starts and ends with a free vertex

Return M

$O(\sqrt{V}) \quad O(|E|)$

$T_W \Rightarrow T(n) = O(\sqrt{V} |E|)$

Q-2)

```
def isBalanced(self) -> bool :
    if self.root:
        res = self._isBalanced(self.root)
        if res == -1:
            return False
        else :
            return True
    else:
        return True

3 usages
def _isBalanced(self, cur_node) -> int :
    if cur_node:
        if not cur_node.left and not cur_node.right:
            return 0
        else:
            left = self._isBalanced(cur_node.left)
            right = self._isBalanced(cur_node.right)

            if left == -1 or right == -1:
                return -1
            if abs(left - right) > 1:
                return -1
            else:
                return max(left, right) + 1
    else:
        return 0
```

$\theta(1)$

$T(n/2)$

$T(n/2)$

$\theta(1)$

$T(n) = 2T(n/2) + \theta(1)$

with master teorem

$a = 2 \quad a \quad b^d$
$b = 2 \quad 2 > 2^0$
$d = 0$

So

$T(n) = \theta(n^{\log_2 2})$

$T(n) = O(n)$

```
1 usage
def height(self) -> int :
    if self.root:
        return self._height(self.root)
    else:
        return 0
3 usages
def _height(self, cur_node) -> int :
    if cur_node:
        if not cur_node.left and not cur_node.right:
            return 0
        else:
            left = self._height(cur_node.left)
            right = self._height(cur_node.right)
            return max(left, right) + 1
    else:
        return 0
```

$\theta(1)$

$T(n/2)$

$T(n/2)$

$\theta(1)$

$T(n) = 2T(n/2) + \theta(1)$

with master teorem

$a = 2 \quad a \quad b^d$
$b = 2 \quad 2 > 2^0$
$d = 0$

So

$T(n) = \theta(n^{\log_2 2})$

$T(n) = O(n)$