1.

a) $T(n) = 3T(n-1) - 2T(n-2) \rightarrow r^2 - 3r + 2r = 0$

$r_1 = 1$ $\quad T(n) = c_1 . 1^n + c_2 . 2^n \quad 2c_1 + c_2 = 2$

$r_2 = 2$

$T(n) = \Theta(2^n) \quad c_1 = 1 \quad c_2 = 2$

b) $T(n) = T(n/2) + 1$; forward substution

$T(1) = 1 \qquad 2^k = n \quad T(n) = \Theta(\log n)$

$T(2) = 1+1$

$T(4) = 1+1+1 \qquad k = \log_2 n$

$T(2^k) = \underbrace{1+1+1 \ldots}_{2^k}$

c) $T(n) = 4T(n-1) - 4T(n-2) + 3n \rightarrow r^2 - 4r + 4 \quad T(n) = c_1 . 2^n + c_2 . 2^n . n + O(n)$

$(r-2)^2$

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

d) $T(n) = 4T(n/2) + n^2$; Backward Substution $\quad 2^k = n \quad a\left(\frac{r^n - 1}{r-1}\right)$

$T(n) = 4[4T(n/4) + n^2] + n^2 = 16T(n/4) + (n^2) + n^2 \quad k = \log_2 n$

$T(n) = \Theta(n^2 \log n)$

$T(n) = 4^k T(n/2^k) + n^2 + n^2 \ldots 4^{k-1} n^2 \rightarrow 4^{\log_2 n} + n^2(1 + 4 \ldots 4^{\log_2 n - 1}) \rightarrow n^2 + n^2 \log n$

e) $T(n) = 2T(n/2) + O(n)$; Master's Theorem

$a = 2 \quad b = 2 \qquad \log_2 2 = 1 = k \rightarrow O(n^k \log^{p+1} n) \rightarrow \Theta(n \log n) = T(n)$

$n^1 \rightarrow k = 1 \qquad p = 0$

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

$T(n) \leq 2T(n/4) + n$

$n^{\log_4 2} \rightarrow \sqrt{n} \qquad f(n) > \sqrt{n} \qquad T(n) = \Theta(n)$

g) $T(n) = T(n/2) + n$: Master's Theorem

$a = 1$ $\qquad \log_2 1 < k$

$b = 2$ $\qquad p \geq 0$ so $\Theta(n^k \log^p n) \to \Theta(n)$

$k = 1$

$p = 0$

h) $T(n) = 2T(\sqrt{n}) + 1$ Backword substution

$T(n) = 2(2T\sqrt{\sqrt{n}} + 1) + 1 \to 4T(\sqrt{\sqrt{n}}) + 2 + 1$

$\vdots$

$T(n) = 2^k T(\sqrt[2^k]{n}) + \underbrace{2^{k-1} + 2^{k-2} \cdots 1}_{2^{\log n}}$

$2^k = n$

$k = \log_2 n$ $\qquad 2^{\log_2 n} \cdot 1 + 2^{\log_2 n} = n + n = 2n \to T(n) = \Theta(n)$

2.

Pseudocodes:

a)

```
def is_balanced(node):
        if node is empty
            return True
        left_height = height_of_tree(node.left)
        right_height = height_of_tree(node.right)
        if ( abs( left_height - right_height ) ≤ 1 and is_balanced(node.left) and is_balanced(node.right))
            return True
        return False
```

b)

```
def height_of_tree(node):
        if node is empty
            return 0
        else:
            return 1 + max( height_of_tree(node.left) , height_of_tree(node.right) )
```

$a \rightarrow \quad T(n) = 2T(n/2) + 1$ $\qquad b \rightarrow \quad T(n) = 2T(n/2) + 1$

$T(n) = 2[2.T(n/4) + 1] + 1$

$\vdots$

$2^k . T(n/2^k) + 1 + 2 ... 2^{k-1}$

$2^k = n \; , \; k = \log n \qquad T(1) = 1$

$so \rightarrow n + \log(n) \rightarrow \Theta(n)$

a algorithm uses b algorith and just does constant if check. So both of their average time complexities are $\Theta(n)$.

3. This problem can ve solved by describing each algorithm in recurrence relations and analyzing them.

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

by implementing Master's Theorem we get $\underbrace{\Theta(n^3)}$ $(a=5 \quad b=2 \quad k=3 \quad p=0)$

$O(n^k \log^p n)$

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

by implementing Master's Theorem we get $O(n \cdot 2^{n/2})$ $(a=2 \quad b=2 \quad f(n)=n)$

$O(f(n) \cdot a^{n/b})$

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

by implementing Master's Theorem we get $\Theta(n^2)$ $(a=3 \quad b=2 \quad k=2 \quad p=0)$

$O(n^k \log^p n)$

We clearly see that the best algorithm is C algoritm. $n^2 < n^3 < n \cdot 2^{n/2}$

4.

Hopcroft karp can be used for this problem.

```
def hopcroft_karp(graph, N):
    match = [N] * (N+1)
    dist = [0] * (N+1)
    matching = 0
    while BFS(graph, match, dist, N):
        for v in range(N)
            if match[v] == N and DFS(graph, match, dist, v, N):
                matching += 1
    return matching
```

Worst Case: $O(E\sqrt{V})$ E: number of edges V: number of vertices.

Best Case: $O(E)$

Average Case: $O(E\sqrt{V})$

5. Number of characters printed is actually the same with n algorithm complexity because it prints 1 character in 1 time. Recurrence relation for this proplem is: $T(n) = 2.T(n/2) + n$ $(a=2 \quad b=2 \quad k=1 \quad p=0)$

by Master's theorem we say that the time complexity is $\Theta(n.\log n)$

So we can say that algorithm prints $n.\log n$ characters.