The Master Method:

$T(n) = aT(\frac{n}{b}) + f(n)$, $a \geq 1$ and $b > 1$

1-) $c < \log_b^a \Rightarrow T(n) = \Theta(n^{\log_b^a})$

2-) $c = \log_b^a \Rightarrow T(n) = \Theta(n^c \log n)$ } If $f(n) = \Theta(n^c)$

3 $c > \log_b^a \Rightarrow T(n) = \Theta(f(n))$

Q-1) a-) $T(n) = \underbrace{27}_{a} T(n/\underbrace{3}_{b}) + n^{\overset{\rightarrow c}{2}}$

$\Rightarrow \log_3^{27} = \log_3^{3^3} = 3$ and $3 > c = 2$ so case 1 is valid for the situation. $\boxed{T(n) = \Theta(n^3)}$

b-) $T(n) = 9T(n/4) + n$

$\log_4^9 \cong 1.58$ and $c = 1$, so; $1.58 > 1$, case 1

$\boxed{T(n) = \Theta(n^{1.58})}$

c-) $T(n) = 2T(n/4) + \sqrt{n}$

$\log_4^2 \cong 0.5$, $c = \frac{1}{2}$ so; $0.5 = 0.5$, case 2

$\boxed{\Rightarrow T(n) = \Theta(n^{0.5} \log n)}$

d-) $T(n) = 2T(n/2) + 17$

$\log_2^2 = 1$, $c = 0$ so; $1 > 0$, case 1

$\boxed{\Rightarrow T(n) = \Theta(n)}$

e-) $T(n) = 2T(\sqrt{n}) + 1$

1) leTs assume that $\boxed{n = 2^k}$ (1) then the expression transform to

$$T(2^k) = 2T(2^{k/2}) + 1$$

2) lets assume that $T(2^k) = S(k)$ so;

$$\boxed{T(2^k) = S(k) = 2S(\tfrac{k}{2}) + 1} \rightarrow \text{use master theorem;}$$

$$\log_2^2 > 0 \Rightarrow \boxed{\Theta(k) = S(k)} = T(2^k) = T(n)$$

→ We need to convert back to the expression from k to n.

if $n = 2^k$ then $\underline{k = \log n}$ so the $\boxed{\Theta(k) = \Theta(\log n)}$

---

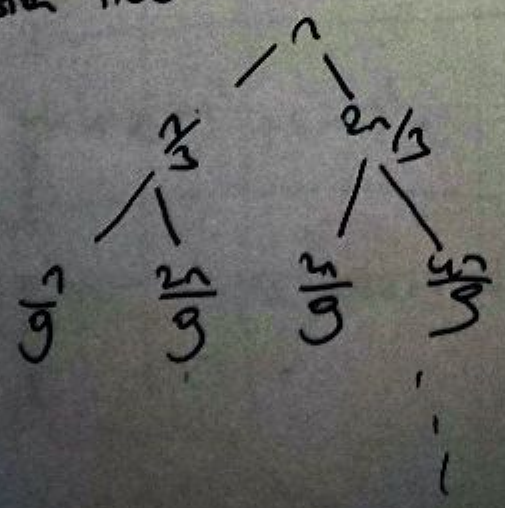
f-)

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

$$\log_2^4 > 1^k \Rightarrow T(n) = \Theta(n^2)$$

g-) $T(n) = T(n/3) + T(2n/3) + O(n)$

Recursion tree:



(total equal to n)

(total = n)

$\log_3 n$ level

→ left most node represents best case and rightmost represents worst case.

g- cont.) $\Rightarrow$ Total value on each level is n and there are $\log_3 n$ level to reach end of the tree. [redacted] Thus;

$$T(n) \geq n \cdot \log_3 n = \Omega(n \log n)$$

$\Rightarrow$ there are $\log_{3/2} n$ level on the worst case and n subproblem on the each level of the worst case.

$$T_n \leq n \cdot \log_{3/2} n = O(n \log n)$$

So; if Ω and O is the same $\boxed{T(n) = \Theta(n \log n)}$

h-)

$$T(n) = T(n-1) + n^c$$

So $\Rightarrow$ $T(n) = n^c + (n-1)^c + (n-2)^c + \cdots + 1$

See the Faulhabers formula. This equation is equal to n^{c+1} on the worst case.

for example; let say $c = 1$ the equation equals to;

$$n^1 + (n-1)^1 + (n-2)^1 + \cdots + 2^1 + 1^1 = \frac{n \cdot n-1}{2} = \frac{n^2 - n}{2} = \Theta(n^2)$$

If the c is equal 1 then the [redacted] recurrence equals to

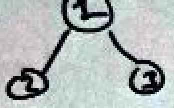
$$\boxed{\Theta(n^{c+1})}$$

i-) $T(n) = T(n-1) + c^n$

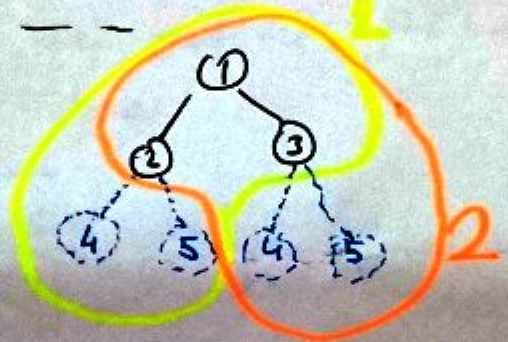
$T(n) = c^n + c^{n-1} + c^{n-2} + \dots + c^2 + c^1 + c^0 = \frac{c^{n+1} - 1}{c-1}$

this relation equals to c^{n+1} on both worst and best case so $\boxed{\Theta(c^{n+1}) = T(n)}$

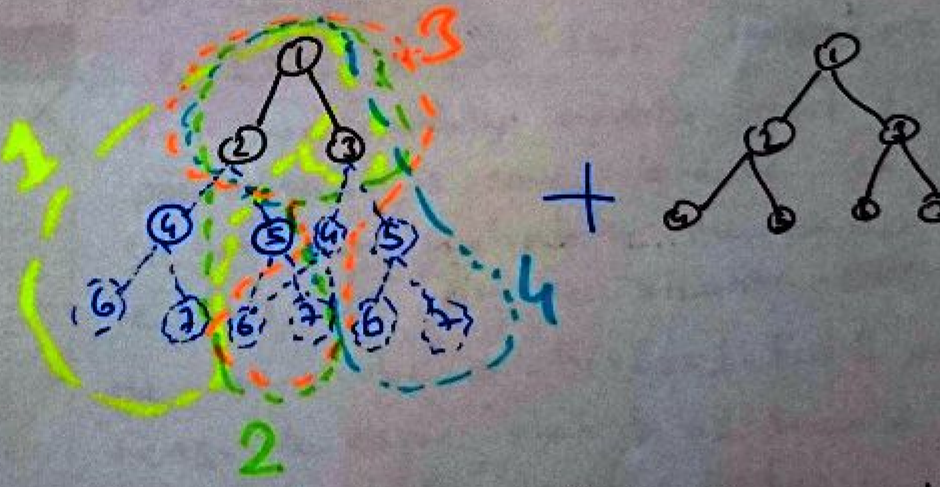
$\boxed{2\text{-}Q}$ a-) $B_3 = 1$ because we can construct only one full tree with 3 vertex



$B_5 = 2$



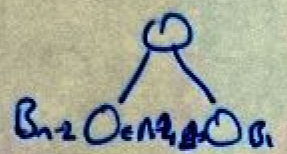
$B_7 = 5$



* There couldn't be any full tree with even numbers of vertices because Full tree states for "each node must has 0 or 2 childs." and that means there must be k+1 vertices in the tree where $\boxed{k \% 2 = 0}$. If a number can be divided 2 without any remainder; that means the number is even and any odd number is equal to adding 1 to any even number. So k+1 is a odd number thats why we cannot use even numbers to calculate Bn.

2-b) We need to consider combination logic to calculate number of binary trees with n vertices.

→ lets assume that we have n vertices;



B_{n-2} ⟵ $n-2, 1$ ⟶ B_1 → $n-1$ vertices

B_{n-4} ⟵ $n-4, 3$ ⟶ B_3 → $n-3$ vertices.

⋮ ⋮

B_3 ⟵ $3, n-4$ ⟶ B_{n-4}

B_1 ⟵ $1, n-2$ ⟶ B_{n-2}

* We can divide the vertices into two subclass (branch) as shown on the illistration below. So; if we sum all the possibilities then we get the total number of possible sub-trees (full binary trees).

→ E.g. lets say $n=9$ → B_9;

B_7	7,1	B_1
B_5	5,3	B_3
B_3	3,5	B_5
B_1	1,7	B_7

8 vertices

→ for [7, 1]; it means that there are B_7 left full tree possibility and also B_1 right full tree possibilies. It is some kind of cartesian product so we can calculate the number of possible full trees for the situation by multiply the two value as $B_1 \times B_7 = 1 \cdot 5 = 5$ possible full trees. We assume that $B_1 = 1$ because 1 vertices with zero child is also a full tree.

→ do the process for each situation;

$B_7 \times B_1 = 5$

$B_5 \times B_3 = 2$

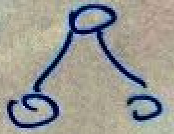
$B_3 \times B_5 = 2$

$B_1 \times B_7 = 5$

$= 14$

2-b-cont.)

→ lets calculate a familiar B_n like B_7:



$B_5 \leftarrow 5,1 \rightarrow B_1$ $= 2\times1 = 2$

$B_3 \leftarrow 3,3 \rightarrow B_3$ $\Rightarrow$ $= 1\times1 = 1$

$B_1 \leftarrow 1,5 \rightarrow B_5$ $= 1\times2 = 2$

$= \underline{\underline{5}}$

⇒ So we can formulate the relation as;

$$B_n = (B_{n-2} \times B_1) + (B_{n-4} \times B_3) + \dots (B_1 \times B_{n-2})$$

$$\boxed{= \sum_{k=1}^{n-2} B_k \times B_{n-k-1}, \text{ for } \forall k, k \text{ is odd.}}$$

2-c) Unfortunately this question is wrong.

3-)

a-)

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

a = Count of the sub-problems.

$\frac{n}{b}$ = Size of the sub-problems.

a-) "dividing into 7 subproblems with one-third of the size then combining them in quadratic time"

$$T(n) = 7T\left(\frac{n}{3}\right) + n^2$$

by master theorem;

$\log_3 7 \cong 1.77$ and $c = 2$

so; $\log_3 7 < 2 \Rightarrow \boxed{T(n) = \Theta(n^2)} = O(n^2)$

Q3-b) $T(n) = T(n-1) + n$

$T(n-1) = T(n-2) + n-1$

$T(n-2) = T(n-3) + n-3$

$\vdots$

$T(1) = T(0) + 1$

$T(n) = T(0) + (1+2+\dots+n)$ → T(0) is a constant.

$T(n) = T(0) + \left(\frac{n \cdot n+1}{2}\right) = \frac{n^2+n}{2} + T(0)$

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

3-c) $T(n) = 4T(\frac{n}{2}) + n^2$

Masters theorem;

$\log_2^4 = 2 = c=2$ so; $T(n) = \Theta(n^2 \log n) =$ $\boxed{= O(n^2 \log n)}$

⇒ So we need to compare the algorithms;

$O(n^2) = O(n^2) < O(n^2 \log n)$

(1) (2) (3)

$\lim_{n \to \infty} \frac{n^2 \log n}{n^2} = \infty$

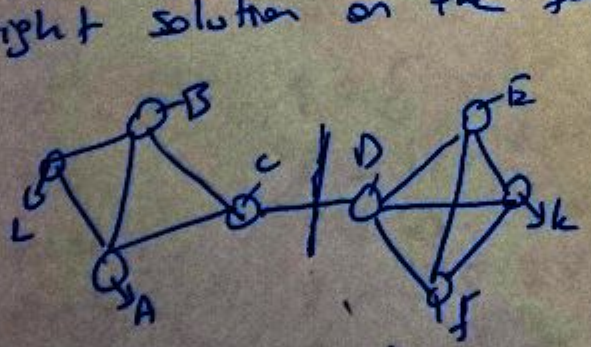
* So we can pick both 1 and 2 algorithm.

Q4) There is no polynomial-time algorithm to solve the min-cut problem <u>exactly</u>. The question asks to give an polynomial-time algorithm, it doesn't ask the optimal solution for the problem because the problem is np-complete and therefore there is no polynomial-time algorithm for the solution of the problem.

* <u>Any offered algorithm which has polynomial running time will be considered as a correct answer.</u>

An example solution for the problem:

(*) We can offer an greedy algorithm to solve the problem in polynomial-time. Our greedy condition [redacted] consider the vertex degrees to solve the problem. Our solution [redacted] wants to find smallest degree vertex and accept the vertex as set one and V− the vertex is the second set. This solution doesn't guarantee the optimal solution but still it is a reasonable solution for our problem. [redacted] E.g. the algorithm can not find the right solution on the following example.



* The algorithm finds vertex "L" [redacted] as the vertex with smallest degree and provides a solution like:

$S_1\{L\}$, $S_2\{A, B, C, D, E, F, K\}$

→ But the optimal solution divides the graph into two set by cutting the edge between C and D.

⇒ Optimal solution only cuts 1 edge but our solution cuts 2 edge at least.

) The code is very similar to an imple-
entation of Catalan numbers but they are not the
me. When we consider the relation between
and number of prints we get some relati-
on like the followings:

$n=0$ print count $=1$

$n=1$ " " $=1$

$n=2$ " " $=5=$

$n=3$ " " $=15=3\times P(2)$

$n=4$ " " $=45=3\times P(3)=3\times 3\times P(2)$

$n=5$ " " $=135=3\times P(4)=3\times 3\times P(3)=3\times 3\times 3\times P(2)$

$n=6$ " " $=405=3\times 3\times 3\times 3\times P(2)$

$\vdots$

$n=n$ " " $=3^{n-2}\times P(2)$

$\Downarrow$

$$T(n)=P(2)\times 3^{n-2}=\frac{5}{9}\cdot 3^n$$

(1) for all $k<\frac{5}{9}$; $k\cdot 3^n<\frac{5}{9}3^n$

$$\boxed{\text{So } T(n)=\Omega(3^n)}$$

(2) for all $k\geq\frac{5}{9}$; $k\cdot 3^n>\frac{5}{9}3^n$

$$\boxed{\text{So } T(n)=O(3^n)}$$

Since $T(n)=\Omega(3^n)$ and $T(n)=O(3^n)$

$$\boxed{T(n)=\Theta(3^n)}$$