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

$T(n)$ is a increasing function

$a = 1 \quad b = 2 \quad d = 0$

$\boxed{b^d = 1}$

$a = b^d \implies T(n) \in \Theta(n^d \log n)$

$\boxed{T(n) \in \Theta(\log n)}$

CamScanner ile tarandı

1) b) While our last algorithm solves the problem with $\Theta(n)$ complexity, first algorithm solves it in $\Theta(n^3)$. On the other hand, last algorithm uses $\Theta(n)$ space, while the first one uses $\Theta(1)$. Dynamic programming type solution is better.

CamScanner ile tarandı

2) $T(n) = 2T(\frac{n}{2}) + 1$ $\quad T(n)$ is a increasing function.

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

$\boxed{b^d = 1 < a = 2}$ $\boxed{b^d < a}$ $\rightarrow T(n) \in \Theta(n^{\log_b a})$

$\boxed{T(n) \in \Theta(n)}$

3)

$$T(n) = \overbrace{F(n) + F(n-1) + \dots + F(n-k+1)}^{k \text{ number of recursive call}}$$

$$F(n) = \sum_{0}^{n} 1 = \boxed{n \in \Theta(n)}$$

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

4) $T(n) = 2T(\frac{n}{2}) + n$ $\quad T(n)$ is a increasing function

$a=2$
$b=2$
$d=1$

$\boxed{a = b^d} \Rightarrow T(n) \in \Theta(n^d \log n)$

$\boxed{T(n) \in \Theta(n \log n)}$

CamScanner ile tarandı

5) Brute Force:

$\boxed{n = \text{power}}$

$$T(n) = \sum_{0}^{n-1} 1 = n \in \Theta(n)$$

Divide and conquer:

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

$a = 1$
$b = 2$
$d = 0$

$\boxed{a = b^d} \Rightarrow \Theta(n^d \log n) = \Theta(\log n)$

$T(n) \in \Theta(\log n)$

CamScanner ile tarandı