CSE321
HW3

1) $T(n) = 2T(n-1) + k$ → Constant time operations count

$T(1)$ will be k

$T(2) = 2T(1) + k$

$T(3) = 2(2T(1) + k) + k = 2^2 T(1) + 2k + k$

$T(4) = 2(2^2 T(1) + 2k + k) + k$

$= 2^3 T(1) + 2^2 k + 2k + k$

⋮

$T(n) = 2^{n-1} \underbrace{T(1)}_{k} + 2^{n-2} k + 2^{n-3} k + \cdots + 2^0 k$

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

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

- Pseudocode is in files I posted, as Python file named find_max_discount.py.

3-)

<u>Worst Case:</u> Function traverses all possible Permutations of "parts" list. For each part the function calls itself on the remaining Parts, $n*(n-1)*(n-2)*\ldots*1 = n!$ times.
This is because worst case complexity is $O(n!)$

<u>Best Case:</u> If the "Parts" list is empty function doesnt enter the loop and finishes in base case directly.
This is because best case complexity is $O(1)$.

<u>Average Case:</u> If the solution is randomly distrubuted average time complexity is still be $O(n!)$.

- Pseudocode is in files I posted, as Python file named most_efficient_sequence.py.

4-)

- Function traverses all possible combinations of coin denominations for each level of recursion. The total number of recursive calls is determined by the number of coins and the amount we want to achive.
- Therefore time complexity of this function is $\underline{O(n.m)}$, n is the target value, m is the coin denomations count.
- Pseudocode is in files I posted, as Python file named min_coins.py.

5-)

$T(n) = 2T(\frac{n}{2}) + k \longrightarrow$ number of constant time operations $O(1)$

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

$a > 2^0$

Master Theorem $= 2 > 2^0 \quad \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \Theta(n)$