1.
A - Recurrance Formula:

$$t(n) = \begin{cases} c & \text{for } 0 \le n \le 2 \\ n + \max(t(n-2), t(n-3)) & \text{for value } n < 3 \\ \max(t(n-2), t(n-3)) & \text{for value } n < 0 \end{cases}$$

B - Pseudocode:

```
def max_independent_set(array):
    memo = []
    three_back = []
    two_back = []
    one_back = []               #set variables to hold returns and tracking
    if len(array) < 3
        if length = 0
            return empty memo
        if length = 1
            append array[0] to memo and return memo
            or if array[0] < 0 just return empty memo
        if length = 2
            append max (array[0],array[1]) to memo
            or if both negative, return empty memo
    else:                       # all arrays length 3+
        if array[2] is negative
            append three_back to array[0]
            append two back to array[1]
            append one back to array[0]     #since 2 is negative and skipped
            add correct current sums to memo
        if array[0] is negative
            append three_back to array[0]
            append two back to array[1]
            append one back to array[2]
            add correct current sums to memo
        if array[1] is negative, and array[0] and array[2] are not:
            append three_back to array[0]
            append two back to array[1]
            append one back to array[2]
            add correct current sums to memo

    for i in range 3-array length
        if array[i] is not negative
            append memo with max(memo[i-3],memo[i-2])
            if memo[i-3] greater
                store three_back in array temp
                move all back variables one step back
                one_back becomes temp
            else memo[i-2] greater
                store two_back in array temp
                move all back variables one step back
                one_back becomes temp
```

```
        else if array[i] is positive
            append memo with (array[i] + max(memo[i-3],memo[i-2])
            if memo[i-3] greater
                store three_back in array temp
                move all back variables one step back
                one_back becomes temp
            else memo[i-2] greater
                store two_back in array temp
                move all back variables one step back
                one_back becomes temp

    #memo stores the max sum at each location in the array
    #we only need to check the last 2 for what is max, and then return
    #one_back or two_back

    if memo[len(memo-1)] > memo[len(memo-2)]
        return one_back
    else:
        return two_back
```

C - Time Complexity: O(n)
    The code uses one for loop that goes the length of the array passed to it. everything else is if statements running in constant time.

2.
B - Time Complexity O(2^n)
    Every additional element of the set makes it run expentionally longer.

Debrief:
1. About 15 hours this week on the course. The assignment was 8 hours and about 7 on the explorations and third party resources to try and understand the material. I ended up not using the knapsack technique for the first problem because I was not very comfortable with the way it was explained.
2. Moderate+
3. 50% maybe. I understand this specific instance of it, but I dont feel confident in the subject overall.
4. I really with there was some more examples for the problems we cover in the explorations. I feel that one simple look at the problem from a theoretical approach and then one specific example thats more in depth would be helpful.