Assignment: Dynamic Programming & Backtracking

*Note: These problems are to be discussed as part of the Group Assignment. (Check this week’s Group Assignment on Canvas for details).*

*The questions asked in this assignment – code implementation and time complexity of your code should be done individually based on the problem-solving strategy discussed within your group.*

1. **Solve Dynamic Programming Problem and find its optimal solution.**

Given a list of numbers, return a subsequence of non-consecutive numbers in the form of a list that would have the maximum sum. When the numbers are all negatives your code should return []

Example 1: Input: [7,2,5,8,6]

Output: [7,5,6] (This will have sum of 18)

Example 2: Input: [-1, -1, 0]

Output: [0] (This is the maximum possible sum for this array)

Example 3: Input: [-1, -1, -10, -34]

Output: []

* 1. Implement the solution of this problem using dynamic Programming. Name your function **max\_independent\_set(nums)**. Name your file **MaxSet.py**
  2. What is the time complexity of your implementation?
     1. O(n)
        1. Looping to fill the dp table: The loop from index 2 to the last element of nums takes O(n) time. Within each iteration, the code performs constant time operations to calculate the maximum sum at each index.
        2. Tracing back to construct the maximum independent set: The while loop iterates until i becomes 0 or 1, which takes O(n) time in the worst case. Within each iteration, the code performs constant time operations such as appending, reversing the list, and updating the index.

1. **Implement a backtracking algorithm** 
   1. Write the implementation to solve the powerset problem discussed in the exercise of the exploration: Backtracking. Name your function **powerset(inputSet)**. Name your file **PowerSet.py**

Given a set of n distinct numbers return its power set.

Example1 :

Input: [1,2,3]

Output: [[1, 2, 3], [1, 2], [1, 3], [1], [2, 3], [2], [3], []] Example2 :

Input: []

Output: [[]]

Note: An empty set is also included in the powerset.

* 1. What is the time complexity of your implementation?
     1. The time complexity of this implementation is O(2^n), where n is the number of elements in the input set. This is because the algorithm generates all possible subsets, and the number of subsets is 2^n, including the empty set. The backtracking process explores all possible choices for each element, leading to an exponential number of recursive calls.