Group 23: Tashfia Tabassum, Rajveer Singh, Richard Phan, Charles Gilbert

Problem 1: Dynamic Programming Approach

The problem requires finding the maximum sum subsequence from a list of numbers while ensuring that the selected numbers are non-consecutive. This type of problem is best solved using Dynamic Programming (DP).

We define dp[i] as the maximum sum that can be obtained by selecting elements from the subarray nums[0:i] while following the non-consecutive condition.

Recurrence relation:

dp[i] = max(dp[i-1], nums[i] + dp[i-2])

Base Cases:

dp[0] = max(0, nums[0])

dp[1] = max(dp[0], nums[1])

Pseudocode

```
def max_independent_set(nums):
    if not nums:
        return []

    n = len(nums)
    if n == 1:
        return [nums[0]] if nums[0] > 0 else []

    # Initialize DP table
    dp = [0] * n
    dp[0] = max(0, nums[0])
    dp[1] = max(dp[0], nums[1])

    # Compute DP values
    for i in range(2, n):
        dp[i] = max(dp[i-1], nums[i] + dp[i-2])

    # Reconstruct the solution
    result = []
    i = n - 1
```

```
    while i >= 0:
        if i == 0:
            if dp[i] > 0:
                result.append(nums[i])
            break
        if dp[i] == dp[i-1]:  # Skip the element
            i -= 1
        else:
            result.append(nums[i])
            i -= 2  # Move two steps back

    return result[::-1]  # Reverse to maintain order
```

Time Complexity

- The solution uses a single pass for filling the dp array (O(n)).
- Another pass is used for reconstructing the subsequence (O(n)).
- Hence, the overall time complexity is O(n).

Problem 2: Backtracking Approach

The power set problem requires generating all possible subsets of a given set. A backtracking approach is an efficient way to explore all subsets systematically.

Steps of the approach:

1. Start with an empty subset.

2. At each step, either:

   - Include the current element in the subset.

   - Exclude the current element from the subset.

3. Recursively generate all subsets.

Pseudocode

```
def powerset(inputSet):
    result = []

    def backtrack(start, subset):
        result.append(subset[:])   # Append a copy of the subset
        for i in range(start, len(inputSet)):
            subset.append(inputSet[i])
            backtrack(i + 1, subset)
            subset.pop()   # Backtrack to explore other possibilities

    backtrack(0, [])
    return result
```

Time Complexity

- The total number of subsets of a set with n elements is 2^n.
- The function generates and stores all these subsets.
- Since each subset is generated in O(n) time (for copying the list), the total complexity is O(n * 2^n).