Kristin Schaefer

HW4 : Dynamic Programming

Sources:

Problem 1

1. https://stackoverflow.com/questions/4487438/maximum-sum-of-non-consecutive-elements

1. Max Independent Set

a. recurrence formula

$$\text{max\_independent\_set}(n) = \begin{cases} 0 & n <= 0 \\ n & n = 1 \\ \max\{\ \text{max\_independent\_set}(n-1),\ \text{max\_independent\_set}(n-2) + n\ \} & n > 1 \end{cases}$$

b. pseudocode

```
mis_helper( nums[1...n], n, memo{ } ):
    # return value stored in memo if exists
    if n in memo:
        return memo[n]

    # otherwise calculate value and store in memo
    if n < 0:
        return 0
    else:
        memo[n] = max(mis_helper(nums, n-1, memo), mis_helper(nums, n-2, memo) + nums[n])

    # if memo is filled, return memo, which has the max sum of each step (so we can calculate optimal solution)
    # otherwise return memo[n] to recursive call
    if n = len(nums) - 1:
        return memo
    else:
        return memo[n]
```

```
max_independent_set( nums[1...n] ):
    # get max sums for each step of solution from mis_helper( )
    max_sums = mis_helper( nums[1...n], len(nums) - 1, { } )

    solution_set = [ ]
    n = len(nums) - 1

    # get optimal solution / set of numbers which create max sum of nums[1...n]
    while n >= 0:
        if n < 1:
            append max_sums[0] to solution_set
            break
        if n == 1:
            append max(max_sums[0], max_sums[1])) to solution_set
            break

        s1 = max_sums[n] - max_sums[n-1]
        s2 = max_sums[n] - max_sums[n-2]
        if s1 = 0 and s2 = 0:
            n = n - 1
        else if s1 > s2:
            n = n - 1
        else:
            append s2 to solution_set
            n = n - 2

    # reverse solution_set[ ]
    return solution_set.reverse()
```

d. time complexity

In mis_helper( ) we make n calls to mis_helper( ) to fill the memo with the max solution. Then in max_independent_set( ) we determine the optimal solution set in n steps. So we then have 2*n, which is O(n).

Kristin Schaefer

HW4 : Dynamic Programming

2. Power Set

b. time complexity

We have $T(n) = 2T(n-1) + \Theta \Rightarrow \Theta(2^n)$