Github Repo: djcriley/comp257-final-project (github.com)

Question:
You have a certain amount of money available to fund grants. You receive some number of funding proposals, which each have a quality rating and an amount of funding requested(price). You want to fund a subset of grants that maximize the sum of their quality ratings, without going over your budget.

Input:

- n: number of proposals (or length of Q)
- List Q of size n. Q[i] corresponds to a given proposal, I, and is a non-negative integer quality rating.
- List F of size n. F[i] corresponds to a given proposal, I, and is a non-negative interger value.
- budget: integer amount of the money available.

Output:

- maxFundableSubset: A list of indexes of the proposals we chose to fund.

Brute Force: Checking all possibilities using recursion to see if you should include the person or not.

Pseudo code:

```
def brute_maxFundableSubset(Q, F, budget):

    n = length(Q)

    # Declare a list called 'subsets' with an empty list of list
    subsets = [[]]

    For 'i' from 0 to 'n', do:
        Subsets += [subset + [i] for subset in subsets]

    # Declare a variable 'maxTotalQuality' and set it to 0
    maxTotalQuality = 0

    # Declare a list called 'maxFundableSubset' and set it to an empty list
    maxFundableSubset = []

    For each 'subset' in 'subsets', do:
        Declare a variable 'totalQuality' and set it to the sum of 'Q[i]' for each 'i' in 'subset'
```

```
    Declare a variable 'totalFunding' and set it to the sum of 'F[i]' for each 'i' in 'subset'

    If 'totalFunding' is <= 'budget' and 'totalQuality' is > 'maxTotalQuality':
        Set 'maxTotalQuality' to 'totalQuality'
        Set 'maxFundableSubset' to 'subset'

  Return 'maxFundableSubset'
```

Greedy: sort by quality/funding and select highest ratios until no more budget

Pseudo code:

```
def greedy_maxFundableSubset(Q, F, budget):

  proposals = combine Q, F and the index into a list of tuples: (q, f, i)

  Sort 'proposals' by 'quality/funding' in descending order

  Declare a list called 'maxFundableSubset' and set it to an empty list

  For each element 'proposal' in 'proposals', do:
    If 'proposal[1]' is <= 'budget':
      Append 'proposal[2]' to 'maxFundableSubset'
      Decrement 'budget' by 'proposal[1]'

  Sort 'maxFundableSubset' in ascending order

  Return 'maxFundableSubset'
```

Dynamic programming:

Pseudo code:

```
def dp_maxFundableSubset(Q, F, budget):

  n = length(Q)

  Declare a 2D list called 'dp' with 'n+1' rows and 'budget+1' columns, and set each element to 0
```

```
For a variable 'i' from 1 to 'n+1', do:
    For a variable 'j' from 1 to 'budget+1', do:
        If 'F[i-1]' is <= to 'j':
            Set the element at 'dp[i][j]' to the maximum of 'dp[i-1][j]' and 'dp[i-1][j-F[i-1]] + Q[i-1]'
        Else:
            Set the element at 'dp[i][j]' to 'dp[i-1][j]'

Declare a list called 'maxFundableSubset' and set it to an empty list

Declare variables 'i' and 'j' and set them both to 'n'

While 'i' is > 0 and 'j' >  0, do:
    If 'dp[i][j]' == 'dp[i-1][j]':
        Decrement 'i' by 1
    Else:
        Append 'i-1' to 'maxFundableSubset'
        Decrement 'j' by 'F[i-1]'
        Decrement 'i' by 1

Reverse 'maxFundableSubset'

Return 'maxFundableSubset'
```

Time analysis:

Greedy: O(n log n) because it sorts the inputs based on a certain criteria. This is n logn time.

Dynamic Programming: O(n) - because it only iterates over the input 1 time.

Brute Force: O(2^n) - because it creates all possible subsets of proposals which is 2^n time.