```
def find_combination(choices, total):
    """
    choices: a non-empty numpy.array of ints
    total: a positive int

    Returns result, a numpy.array of length len(choices)
    such that
        * each element of result is 0 or 1
        * sum(result*choices) == total
        * sum(result) is as small as possible
    In case of ties, returns any result that works.
    If there is no result that gives the exact total,
    pick the one that gives sum(result*choices) closest
    to total without going over.
    """
    counter = 1
    result = np.array([0 for i in range(len(choices))])
    while len(bin(counter)[2:]) <= len(choices):
        a = np.array(list(map(int, bin(counter)[2:].zfill(len(choices)))))

        if sum(a * choices) <= total:
            if total - sum(a * choices) < total - sum(result * choices):
                result = a
            elif total - sum(a * choices) == total - sum(result * choices):
                if sum(a) < sum(result):
                    result = a
        counter += 1
    return result

#ALTERNATE SOLUTION

def find_combination(choices, total):
    """
    choices: a non-empty list of ints
    total: a positive int

    Returns result, a numpy.array of length len(choices)
    such that
        * each element of result is 0 or 1
        * sum(result*choices) == total
        * sum(result) is as small as possible
    In case of ties, returns any result that works.
    If there is no result that gives the exact total,
    pick the one that gives sum(result*choices) closest
    to total without going over.
    """
    if choices == [] or total == 0:
        result = np.array([0] * len(choices))
    elif choices[0] > total: #cannot afford current item
        # Do not take first item
        result = np.concatenate([[0], find_combination(choices[1:],
total)]).astype('int')
    else:
        # Take first item
        result_with = np.concatenate([[1], find_combination(choices[1:], total -
choices[0])]).astype('int')
        total_value_with = sum(choices * result_with)

        # Do not take first item
        result_without = np.concatenate([[0], find_combination(choices[1:],
total)]).astype('int')
        total_value_without = sum(choices * result_without)

        # Choose better branch
        if total_value_with > total_value_without:
            result = result_with
        elif total_value_with < total_value_without:
            result = result_without
        else:
```

```
            if sum(result_with) < sum(result_without):
                result = result_with
            else:
                result = result_without
    return result
```

```
def lecture_activities(N, aLecture):
    '''
    N: integer, number of trials to run
    aLecture: Lecture object

    Runs a Monte Carlo simulation N times.
    Returns: a tuple of (1) a float representing the mean number of
             days it takes to have a day in which all 3 actions take place
             (2) the total width of the 95% confidence interval around that mean
    '''
    days_list = []
    for trial in range(N):
        days = 1
        while (random.random() > aLecture.get_listen_prob()) or \
              (random.random() > aLecture.get_sleep_prob()) or \
              (random.random() > aLecture.get_fb_prob()) :
            days += 1
        days_list.append(days)
    (mean, std) = get_mean_and_std(days_list)
    return (round(mean, 3), round(1.96*std*2, 3))
```

```
def greedySum(L, s):
    current_sum = 0.0
    multiples = []
    for val in L:
        # find the largest multiple
        multiple = int((s - current_sum) / val)
        current_sum += (val * multiple)
        multiples.append(multiple)

    if current_sum != s:
        return "no solution"
    else:
        return sum(multiples)
```